Universidad Carlos III de Madrid

Departamento de Teoría de La Señal

Leganés, Octubre de 2015

PROYECTO FIN DE CARRERA

ESTUDIO DE LA APLICACIÓN DE LAS METODOLOGÍAS ÁGILES PARA PROYECTOS SOFTWARE EN EL ÁMBITO DE LAS TI

Autor | González del Río, Jorge

Tutora | Pérez Leal, Raquel

Título: Estudio de la aplicación de las metodologías ágiles para proyectos software en el ámbito de las TIC.

Autor: Jorge González del Río

Director: Raquel Pérez Leal

EL TRIBUNAL

Presidente: Harold Molina Bulla

Vocal: Jesús Martín

Secretario: Jesús Arroyo Hernández

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 30 de Octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la **CALIFICACIÓN** de _____

VOCAL

SECRETARIO **PRESIDENTE**

Agradecimientos

Dedico este proyecto a todos aquellos que no me dejan conformarme.

Resumen

Este proyecto parte del origen de la gestión de proyectos con las metodologías denominadas tradicionales para posteriormente describir la filosofía ágil y varias metodologías derivadas de dicha filosofía. Las metodologías elegidas han sido Scrum, eXtrema Programing, Crystal, Kanban y Scrumban. Una vez descritas estas metodologías ágiles se ha planteado una herramienta que sirva como ayuda para poder elegir la metodología ágil que más se ajusta a un proyecto o a una organización. Para ello, se han categorizado las metodologías elegidas en función del cumplimiento de los valores y principios de las metodologías ágiles, de forma que la evaluación de esas mismas categorías para un nuevo proyecto pueda compararse con los valores de cada una de las metodologías y determinar cuál de ellas es la que mejor se adapta a las necesidades. Finalmente se ejemplifica una iteración de un desarrollo software llevado a cabo con metodología Scrum. Se plantean una sería de funcionalidades a desarrollar y se describe cómo el equipo irá completándolas durante tres iteraciones de desarrollo.

Abstract

The project makes an overview of the origin of project management from traditional methodologies to nowadays new methodologies known as agile. The agile methodologies described are Scrum, eXtreme Programing, Crystal, Kanban and Scrumban. After that, the project proposes a tool that helps us to choose which agile methodology is more suitable to a project or an organization. In order to do this, the chosen metodologies have been classified based on the values and principles of agile philosophy, helping to evaluate which of these categories fits best to a new project, when matching with the values of each one of the methodologies we have initially described. In the end, I have carried out and example of a software development project in Scrum methodology. Some features have been presented and I describe how the team completes it in three iterations of development.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Motivación

La elección de esta temática como Proyecto Fin de Grado surge de mi actividad profesional en el ámbito del desarrollo de software. En los últimos años he participado en proyectos gestionados tanto con metodologías predictivas como con metodologías ágiles, comprobando en primera persona como la productividad y la cohesión de los equipos mejora con la aplicación de estas últimas. Recientemente he realizado un curso sobre la metodología SCRUM, obteniendo la certificación de Scrum Master, lo que me ha supuesto una motivación añadida para seguir aprendiendo sobre el tema de la agilidad.

Por todo esto, he considerado beneficioso para mi desarrollo profesional realizar un estudio de las principales metodologías de gestión. Esto me permitirá tener una visión global de las mismas y capacitarme para evaluar y decidir qué opción se adapta mejor a cada proyecto concreto.

Si bien las metodologías tradicionales o predictivas se adaptan perfectamente a entornos poco cambiantes y con bajo nivel de incertidumbre, la realidad de la gran mayoría de los

proyectos software es bien distinta. Las necesidades y prioridades cambian rápidamente, obligando a los equipos a rediseñar constantemente los desarrollos. En estos entornos de requisitos cambiantes es donde se aprecian con mayor claridad las diferencias entre ambos tipos de metodologías. En el caso de las metodologías tradicionales el resultado suele ser un retraso en la entrega del producto final o un sobrecoste del proyecto, lo que trae asociado un descontento por parte del cliente final. Las metodologías ágiles tienen en su ADN la adaptación al cambio como su mejor virtud, y aunque tampoco se puede decir que sean milagrosas, entregarán al cliente las partes del producto con mayor valor en primer lugar, lo que se adapta por completo a la tan de moda filosofía "Just In Time".

Objetivos del proyecto

Este proyecto tiene como objetivo estudiar las principales metodologías ágiles y plantear una sencilla herramienta que ayude a elegir que metodología ágil se adapta mejor a las necesidades de un proyecto u organización. El estudio se centra en los proyectos de desarrollo software, por ser un sector para el que se han demostrado los mayores beneficios de dicha filosofía, aunque esto no quiere decir que todo el proyecto sea extrapolable a otros sectores.

Metodología

Para alcanzar los objetivos planteados se seguirán los siguientes pasos:

- Repaso de la historia de gestión de proyectos.
- Presentación de la filosofía ágil.
- Descripción de las principales metodologías ágiles.
- Clasificación de las principales características de las metodologías ágiles estudiadas.
- Presentación de herramienta de selección de metodologías ágiles.
- Ejemplo de proyecto de desarrollo software realizado mediante metodología Scrum.

Capítulo 2

Metodologías tradicionales de gestión de proyectos

2.1. Definición de proyecto

El desarrollo de productos, la prestación de servicios, o la organización de la propia empresa son trabajos que pueden tomar la forma de proyectos o de operaciones.

En ambos casos se comparten tres características comunes:

- Los realizan personas.
- Para su ejecución se dispone de recursos limitados.
- Se llevan a cabo siguiendo una estrategia de actuación.

La diferencia fundamental entre operaciones y proyectos es que mientras que las operaciones se ejecutan de forma repetitiva para obtener resultados de similares características, los proyectos producen resultados únicos. Se considera proyecto a la ejecución de un trabajo que

además de requerir personas, recursos y ejecución controlada, es un desarrollo único y se desarrolla en un marco temporal preestablecido.

Según [1]: "Un PROYECTO es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único crear un producto, servicio o resultado único".

- "Temporal: significa que cada proyecto tiene un comienzo y final definidos"
- "Productos, servicios o resultados únicos: un proyecto crea productos entregables únicos"

Las construcciones de ingeniería civil como puentes o edificios son ejemplos clásicos de obras realizadas como proyectos, y en general lo es el desarrollo de cualquier sistema singular.

Cada proyecto tiene objetivos y características propias y únicas. Algunos necesitan el trabajo de una sola persona, y otros el de cientos de ellas, pueden durar unos días o varios años.

Algunos ejemplos de proyectos:

- Diseño de un nuevo ordenador portátil.
- Construcción de un edificio.
- Desarrollo de un sistema de software.
- Implantación de una nueva línea de producto en una empresa.
- Diseño de una campaña de marketing.

2.2. Origen de la gestión de proyectos

Los proyectos existen desde siempre. Cualquier trabajo para desarrollar algo único es un proyecto, pero la gestión de proyectos es una disciplina relativamente reciente que comenzó a forjarse en los años sesenta. La necesidad de su profesionalización surgió en el ámbito militar. En los años 50, el desarrollo de grandes proyectos militares requería la coordinación del trabajo conjunto de equipos y disciplinas diferentes en la construcción de sistemas únicos. Bernard Schriever[1], arquitecto de desarrollo de misiles balísticos Polaris es considerado el padre de la gestión de proyectos, porque desarrolló el concepto de "concurrencia" integrando todos los elementos del plan de desarrollo en un solo programa y presupuesto, ejecutándolos en paralelo y no secuencialmente. Consiguió de esta forma reducir considerablemente los tiempos de ejecución. Siguiendo los pasos de la industria militar, la del automóvil también comenzó a aplicar técnicas de gestión de proyectos para la gestión y coordinación de la gestión del trabajo entre áreas y equipos funcionales diferentes. Comenzaron a surgir técnicas específicas, histogramas, cronogramas, los conceptos de ciclo de vida del proyecto o descomposición en tareas (WBS -Work Breakdown Structure).

[1] https://en.wikipedia.org/wiki/Bernard_Adolph_Schriever

En 1960, Meter Norden, del laboratorio de investigación de IBM, en su seminario de Ingeniería de Presupuesto y Control presentado ante American Management Association[2], indicó:

- Es posible relacionar los nuevos proyectos con otros pasados y terminados para estimar sus costes.
- Se producen regularidades en todos los proyectos
- Es absolutamente necesario descomponer los proyectos en partes de menor dimensión para realizar planificaciones.

La solvencia demostrada por la gestión de proyectos en la industria militar, y en la automovilística más tarde, para solucionar los problemas comunes de calidad, fechas y costes coincide en el tiempo con la presión que todas las industrias experimentan en mayor o menor medida para reducir los tiempos de salida al mercado, los costes de producción. Como resultado, empresas de todos los sectores: farmacéuticos, químicos, servicios, tecnologías de la información, etc. adoptan técnicas de gestión de proyectos, y contribuyen al desarrollo de un cuerpo de conocimiento común y único para la gestión de proyectos.

2.3. Fundamentos de las metodologías predictivas

La gestión de proyectos desarrollada en las últimas décadas del siglo pasado se basa en la planificación del trabajo, y en el posterior seguimiento y control de la ejecución. La planificación se realiza sobre un análisis detallado del trabajo que se quiere realizar y su descomposición en tareas. Parte por tanto de un proyecto de obra o requisitos iniciales detallados de lo que se quiere hacer. Sobre esa información se desarrolla un plan adecuado a los recursos y tiempos disponibles, y durante la construcción se sigue de cerca la ejecución para detectar posibles desviaciones y tomar medidas para corregirlas. Se trata por tanto de una gestión "predictiva", que predice a través de un plan inicial las características del desarrollo: tiempos, recursos, costes y secuencia de operaciones. Su principal objetivo es conseguir que el desarrollo se lleve a cabo según lo "previsto" y basa el éxito del proyecto en los tres puntos apuntados: agendas, costes y calidad.

La gestión de proyectos predictiva o clásica es una disciplina formal de gestión basada en la planificación, ejecución y seguimiento a través de procesos sistemáticos, repetibles y escalables.

- Establece como criterios de éxito: calidad, tiempo y costes.
- Asume que el proyecto se desarrolla en un entorno estático y predecible.

[2] http://www.amanet.org/advantage/About-Us.aspx

- El objetivo de su esfuerzo es mantener el cronograma, el presupuesto y los recursos.
- Divide el desarrollo en fases a las que considera "ciclo de vida", con una secuencia de tipo: Concepto, requisitos, diseño, planificación, desarrollo, cierre.

2.4. Principales metodologías predictivas

El desarrollo de sistemas complejos que requerían el trabajo conjunto y sincronizado de varias disciplinas o ingenierías hizo evidente en los años 60 la necesidad de desarrollar métodos de organización y de trabajo para evitar los problemas que se repetían con frecuencia en los proyectos:

- Desbordamiento de agendas.
- Desbordamiento de costes.
- Calidad o utilidad del resultado obtenido.

Para dar respuesta a esta necesidad, a partir de los años 60 surgieron organizaciones que han desarrollado el cuerpo de conocimientos y las prácticas necesarias para gestionar esos trabajos con las mejores garantías de previsibilidad y calidad de los resultados. Ese conocimiento se ha ido desarrollando y configurando como la base de una nueva profesión cuya finalidad es el éxito de los proyectos: La gestión de proyectos. Las organizaciones más relevantes en esta línea son:

- Internacional Project Managenet Association (IPMA[3]), fundada en 1965
- Project Management Institute (PMI[4]) constituido en 1965
- Más tarde surgió Prince2[5], que comenzó a trabajar en 1989.

Organizaciones para desarrollar metodologías, o metodologías alrededor de las que se crean organizaciones.

IPMA y PMI surgieron como organizaciones profesionales para el desarrollo de cuerpos de conocimientos, metodologías y procesos para la gestión de proyectos. Prince2 ha tenido la evolución inversa. Comenzó siendo una metodología, alrededor de la que se ha terminado creando una organización.

También en este sentido la evolución ha sido diferente para Prince2. PMI e IPMA tuvieron desde el principio como finalidad el desarrollo de un conocimiento de gestión válido para cualquier proyecto. Sin embargo, Prince2 comenzó siendo un modelo de referencia para proyectos específicos de Tecnologías de la Información, desarrollado por la Central Computer and Telecommunications Agency (CCTA) del Gobierno Británico; y a partir de una revisión llevada a cabo en 1996 se decidió ampliar su ámbito de validez, para cualquier tipo de proyecto.

[3] http://ipma.ch/
[4] http://www.pmi.org/
[5] https://www.prince2.com/europe/training

Capítulo 3

Metodologías ágiles

3.1. Origen de las metodologías ágiles

A mediados de los años 90 comenzó a forjarse una definición moderna de desarrollo ágil del software como una reacción contra las metodologías predictivas utilizadas hasta el momento, consideradas excesivamente pesadas y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo. En el año 2001 diecisiete miembros destacados de la comunidad software, incluyendo algunos de los creadores o impulsores de las metodologías de software, se reunieron en Utah (Estados Unidos) y adoptaron el nombre de "Metodologías ágiles" para denominar a esta nueva corriente de desarrollo iterativo e incremental donde las soluciones y requisitos vienen dados por equipos colaborativos y multidisciplinares. Poco después, algunos de estos miembros formaron la conocida como "Alianza ágil"[6], una organización sin ánimo de lucro que promueve el desarrollo ágil de aplicaciones. Desde ese momento hasta la actualidad las metodologías ágiles han ido adquiriendo gran auge dentro de la

[6] http://www.agilealliance.org/

industria software y las organizaciones más punteras ya apuestan por este nuevo enfoque para desarrollar sus productos.

3.2. Introducción al modelo ágil. El por qué de las metodologías ágiles

En la gestión de proyectos se pueden aplicar dos tipos de metodologías, predictivas o ágiles, siendo los factores para llevar a cabo el proyecto lo que haga en cada proyecto decantarse por unas u otras. El fin con el que se lleva a cabo el proyecto es uno de los factores más importantes para diferenciar las metodologías predictivas de las ágiles. En el caso de las metodologías predictivas le dan una mayor importancia a los procesos que se deben realizar para finalizar el proyecto, mientras que en las metodologías ágiles le dan una mayor importancia a la utilidad o resultado a obtener.

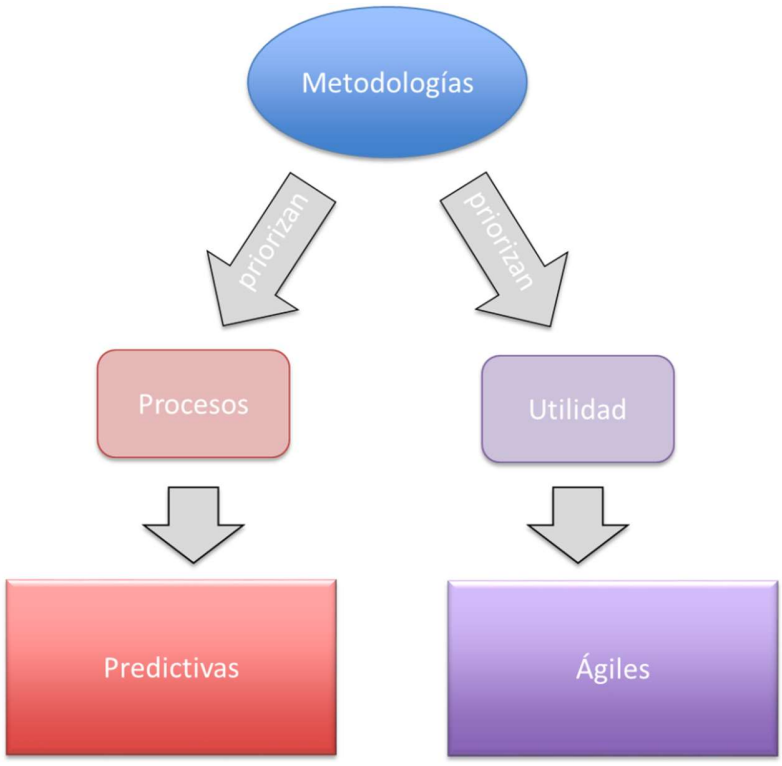


Ilustración 1. Metodologías ágiles vs predictivas

Mientras que en las metodologías predictivas se asume un entorno estable y predecible, la realidad es que en un alto porcentaje de proyectos, y muy especialmente en el desarrollo de software, el dinamismo y la variabilidad del mercado muestran dichas metodologías como rígidas y poco adaptables a las necesidades de dichos proyectos. En las metodologías predictivas se consideran que los proyectos tienen un comportamiento regular, guiado y que se encuentra dentro de un marco estático y predecible de factores (acciones y necesidades del entorno). Además, se establecen los procesos como la unidad mínima consecución del proyecto, identificándolos en términos de tiempo, coste y recursos y estableciendo una planificación y control sobre ellos. Sin embargo, el mercado obliga al rápido desarrollo de aplicaciones con un ciclo de vida cada vez más corto y que las necesidades y acciones queden reflejadas en el proceso de construcción del proyecto.

En este entorno inestable, que tiene como factor inherente el cambio y la evolución rápida y continua, la ventaja competitiva se encuentra en aumentar la productividad y satisfacer las variantes necesidades del cliente en el menor tiempo posible para proporcionar un mayor valor al negocio. Las metodologías convencionales presentan diversos problemas a la hora de abordar un amplio rango de proyectos industriales en este tipo de entornos. Entre estos problemas podemos destacar los siguientes: Perciben la captura de requisitos del proyecto como una fase previa al desarrollo del mismo que, una vez completada, debe proporcionar una visión clara y definida de qué desea el cliente. Se trata de evitar a toda costa que se produzcan cambios en el conjunto de requisitos inicial, puesto que a medida que avanza el proyecto resulta más costoso solucionar los errores detectados o introducir modificaciones, y pretenden delegar toda responsabilidad económica en el cliente en caso de que estos cambios de requisitos se produzcan. Por este motivo, se les conoce también como metodologías predictivas. Sin embargo, el esfuerzo, tanto en coste como en tiempo, que supone hacer una captura detallada de todos los requisitos de un proyecto al comienzo del mismo es enorme y aplicarla depende del marco del proyecto y sus factores. Además, en muchas ocasiones el cliente no conoce sus propias necesidades con la profundidad suficiente como para definirlas de forma exacta a priori y, a menudo, estas necesidades y sus prioridades varían durante la vida del proyecto. Establecer mecanismos de control es una de las opciones existentes para protegerse de estos cambios, aunque frecuentemente provocan la insatisfacción de los clientes, que perciben el desarrollo del proyecto como algo inflexible que no se adapta a sus necesidades y que si lo hace repercute negativamente en costes añadidos al presupuesto del proyecto. Por otro lado, en muchas ocasiones el proceso de desarrollo convencional está oprimido por excesiva documentación no siempre útil. Un porcentaje elevado del tiempo de desarrollo de un producto software se dedica o, desde el punto de vista de las metodologías ágiles, se malgasta en crear documentación que finalmente no se utiliza y que, por tanto, no aporta valor al negocio. Además, esta documentación innecesaria entorpece las labores de mantenimiento de la propia documentación útil lo que provoca que en muchas ocasiones el mantenimiento de la documentación se obvie agudizando, de este modo, el coste en las tareas de documentación futuras. Evidentemente, estas circunstancias no se adaptan a las restricciones de tiempo del mercado actual. Otra dificultad añadida al uso de metodologías convencionales es la lentitud del proceso de desarrollo. Es difícil para los desarrolladores

entender un sistema complejo en su globalidad lo que provoca que las diferentes etapas del ciclo de vida convencional transcurran lentamente, ya que identificar los procesos como las unidades mínimas a desempeñar, como es el caso de las metodologías predictivas, hace que se deba finalizar de manera secuencial los procesos, es decir, uno después de otro, sin opción a modificarlos en su transcurso. Dividir el trabajo en tareas abordables ayuda a minimizar los fallos y, por tanto, el coste de desarrollo. Además, permite liberar funcionalidad progresivamente, según indiquen los estudios de las necesidades del mercado que aportan mayor beneficio a la organización. En la feroz competencia del mercado vigente, en la que los productos quedan obsoletos rápidamente, se pide básicamente rapidez, calidad y reducción de costes, pero para asumir estos retos, es necesario tener agilidad y flexibilidad.

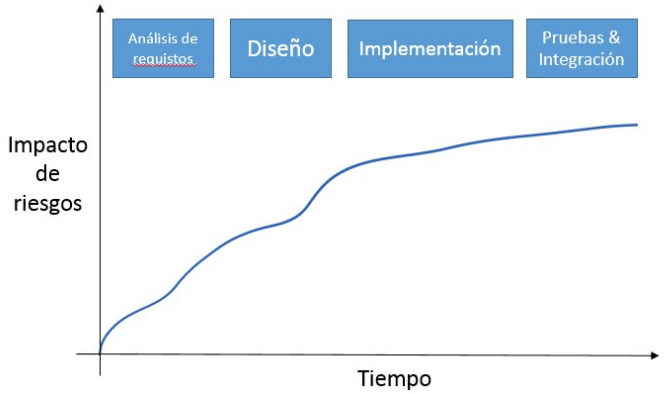


Figura 1. Distribución del riesgo en un desarrollo en cascada

Asimismo, las metodologías convencionales tienden a acumular los riesgos y dificultades que surgen en el desarrollo del producto al final del proyecto, como puede apreciarse en la Figura 1, repercutiendo en retrasos en la entrega de productos o influyendo en la incorrecta ejecución de las últimas fases del ciclo de vida.

En contraposición a las metodologías convencionales, las metodologías ágiles aparecen como alternativa atractiva para adaptarse a este entorno. Son apropiadas cuando los requisitos son emergentes y cambian rápidamente. De este modo, presentan diversas ventajas en el contexto actual:

- Capacidad de respuesta a cambios a lo largo del desarrollo ya que no los perciben como un lastre sino como una oportunidad para mejorar el sistema e incrementar la satisfacción del cliente, considerando la gestión de cambios como un aspecto característico del propio proceso de desarrollo software.
- Entrega continúa y en plazos breves de software funcional lo que permite al cliente verificar in situ el desarrollo del proyecto, ir disfrutando de la funcionalidad del producto

progresivamente y comprobando si satisface sus necesidades, mejorando de esta forma su satisfacción. Además, el desarrollo en ciclos de corta duración favorece que los riesgos y dificultades se repartan a lo largo del desarrollo del producto, principalmente al comienzo del mismo y permite ir aprendiendo de estos riesgos y dificultades (ver Figura 2).

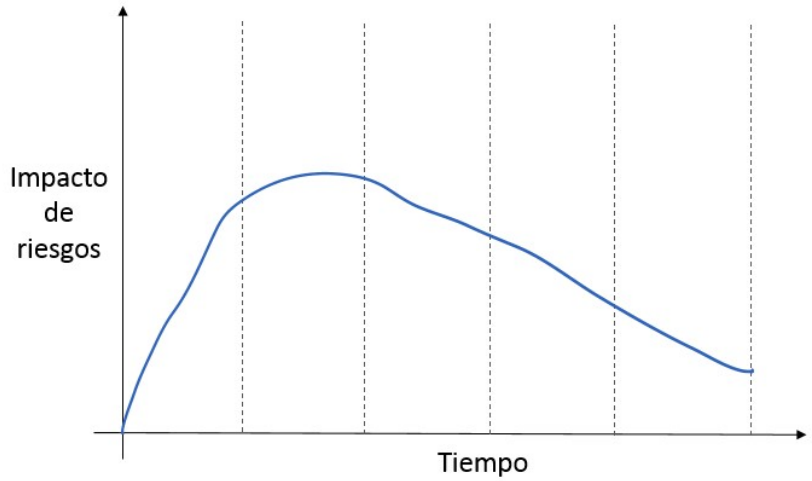


Figura 2. Distribución del riesgo en un desarrollo ágil

- Trabajo conjunto entre el cliente y el equipo de desarrollo con una comunicación directa que pretende mitigar malentendidos, que constituyen una de las principales fuentes de errores en productos software, y exceso de documentación improductiva.
- Importancia de la simplicidad, eliminando el trabajo innecesario que no aporta valor al negocio.
- Atención continua a la excelencia técnica y al buen diseño para mantener una alta calidad de los productos.
- Mejora continua de los procesos y el equipo de desarrollo, entendiendo que el éxito depende de tres factores: éxito técnico, éxito personal y éxito organizacional.

3.3. El Manifiesto Ágil. Valores y principios

"La agilidad es un comportamiento persistente o habilidad, de entidad sensible, que presenta flexibilidad para adaptarse a cambios, esperados o inesperados, rápidamente; persigue la duración más corta en tiempo; usa instrumentos económicos, simples y de calidad en un ambiente dinámico; y utiliza los conocimientos y experiencia previos para aprender tanto del entorno interno como del externo" [2]. Por tanto, el desarrollo ágil no especifica unos procesos o métodos que seguir, aunque bien es cierto que han aparecido algunas prácticas asociadas a este movimiento. El desarrollo ágil es más bien una filosofía de desarrollo software. El punto de partida se establece en las ideas emanadas del Manifiesto Ágil tras la reunión de Utah [3], un documento que resume la filosofía agile estableciendo cuatro valores y doce principios.

Según el Manifiesto se valora:

Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proceso software. Este primer valor expresa que es preferible utilizar un proceso indocumentado con buenas interacciones personales que un proceso documentado con interacciones hostiles. Se considera que no se debe pretender construir primero el entorno y esperar que el equipo se adapte automáticamente sino al revés, construir primero el equipo y que éste configure su propio entorno. El talento, la habilidad, la capacidad de comunicación y de tratar con personas son características fundamentales para los miembros de un equipo ágil.

Desarrollar software que funcione por encima de una completa documentación. Este valor es utilizado por muchos detractores de las metodologías ágiles que argumentan que éstas son la excusa perfecta para aquellos que pretenden evitar las tareas menos gratificantes del desarrollo software como las tareas de documentación. Sin embargo, el propósito de este valor es acentuar la supremacía del producto por encima de la documentación. El objetivo de todo desarrollador es obtener un producto que funcione y cumpla las necesidades del cliente y la documentación es un artefacto que utiliza para cumplir su objetivo. Por tanto, no se trata de no documentar sino de documentar aquello que sea necesario para tomar de forma inmediata una decisión importante. Los documentos deben ser cortos y centrarse en lo fundamental. Dado que el código es el valor principal que se obtiene del desarrollo se enfatiza en seguir ciertos estándares de programación para mantener el código legible y documentado.

La colaboración con el cliente por encima de la negociación contractual. Se propone una interacción continua entre el cliente y el equipo de desarrollo de tal forma que el cliente forme un tándem con el equipo de desarrollo. Se pretende no diferenciar entre las figuras cliente y equipo de desarrollo sino que se apuesta por un solo equipo persiguiendo un objetivo común.

Responder a los cambios más que seguir estrictamente un plan. Planificar el trabajo a realizar es muy útil y las metodologías ágiles consideran actividades específicas de planificación a corto plazo. No obstante, adaptarse a los cambios es vital en la industria software actual y, por tanto, también consideran mecanismos para tratar los cambios de prioridades.

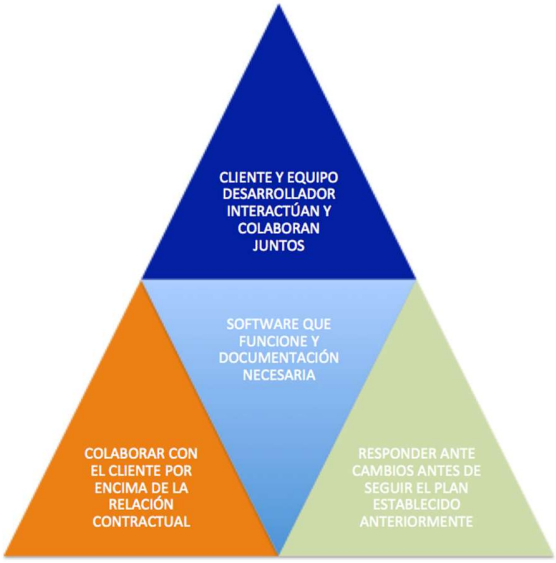


Figura 3. Valores ágiles

Para cumplir estos valores (Figura 3) se siguen doce principios que establecen algunas diferencias entre un desarrollo ágil y uno convencional:

1. La prioridad es satisfacer al cliente mediantes tempranas y continuas entregas de software que le aporten valor.
2. Dar la bienvenida a los cambios de requisitos. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Liberar software que funcione frecuentemente, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. Los miembros del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y apoyo que necesiten y confiar a en ellos para conseguir finalizar el trabajo.
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la principal medida de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se auto organizan.
12. En intervalos regulares el equipo debe reflexionar sobre cómo ser más efectivo y según estas reflexiones ajustar su comportamiento.

Estos principios (Figura 4) marcan el ciclo de vida de un desarrollo ágil, así como las prácticas y procesos a utilizar.

Figura 4. Principios ágiles

3.4. Principales metodologías ágiles

3.4.1. Scrum

3.4.1.1. Origen

Scrum surge a principios de los años 80 a partir del estudio de Ikujiro Nonaka e Hirotaka Takeuchi sobre cómo se desarrollaban los nuevos productos las principales empresas de manufactura tecnológica: Fuji-Xerox, Canon, Honda, Nec, Epson, Brother, 3M y Hewlett-Packard.

El termino Scrum surge de comparar la nueva forma de trabajo en equipo con el avance en formación de scrum de los jugadores de Rugby. Nonaka y Takeuchi realizaron esta comparación en su estudio y el término "scrum" quedo acuñado para referirse a la nueva forma de trabajo.

Posteriormente, en 1995 Ken Schwaber presentó "Scrum Development Process" en OOPSLA 95 [7](Object-Oriented Programming Systems & Application conference), un marco de reglas para desarrollo de software, basado en los principios de scrum, y que él había empleado en el desarrollo de Delphi y Jeff Sutherland en su empresa Easel Corporation.

3.4.1.2. Introducción al modelo

Uno de los principios fundamentales de Scrum consiste en la entrega incremental de valor durante sucesivas iteraciones. De esta forma se fomenta el feedback temprano por parte del cliente. Lo normal en el arranque de cualquier proyecto es que el nivel de incertidumbre sea muy alto y la visión del resultado final no sea lo suficientemente clara. En estas circunstancias, resulta imposible o más bien poco realista preparar una hoja de ruta para conseguir los objetivos. El producto se dividirá en funcionalidades o características que se irán priorizando iteración a iteración tratando de maximizar el valor generado por cada incremento entregado. De esta forma el cliente recibirá las funcionalidades o fragmentos más valiosos para su negocio en las fases más tempranas del proyecto. Sobre estos incrementos recibidos, el cliente podrá ajustar el resultado en las siguientes iteraciones mediante el feedback temprano al equipo encargado de la ejecución del proyecto. Como consecuencia el riesgo disminuirá según avancen las iteraciones al contrario de los proyectos tradicionales en los que el nivel de riesgo se mantiene o incrementa hasta el día de la entrega.

Scrum se basa en la teoría de control de procesos empíricos (empiricism en inglés). Esta teoría afirma que el conocimiento proviene de la experiencia y de las decisiones basadas en lo que se conoce. Scrum emplea un enfoque iterativo e incremental (Sprints) para optimizar la

[7] http://dblp.uni-trier.de/db/conf/oopsla/oopsla95.html

previsibilidad y el control de riesgos. Tres pilares soportan cada aplicación de control de procesos empíricos: Transparencia, Inspección y Adaptación.

- **Transparencia**

Los aspectos significativos de los procesos deben ser visibles para los responsables de los resultados. La Transparencia exige que esos aspectos sean definidos por un estándar conjunto para que los observadores compartan una visión común de lo que se está viendo.

Se debe definir un entendimiento para referirse a un proceso específico y compartirlo con todos los participantes. De forma tal forma que quien realiza una tarea y quien la recibe compartan la misma definición del proceso. Por ejemplo: “Terminado”.

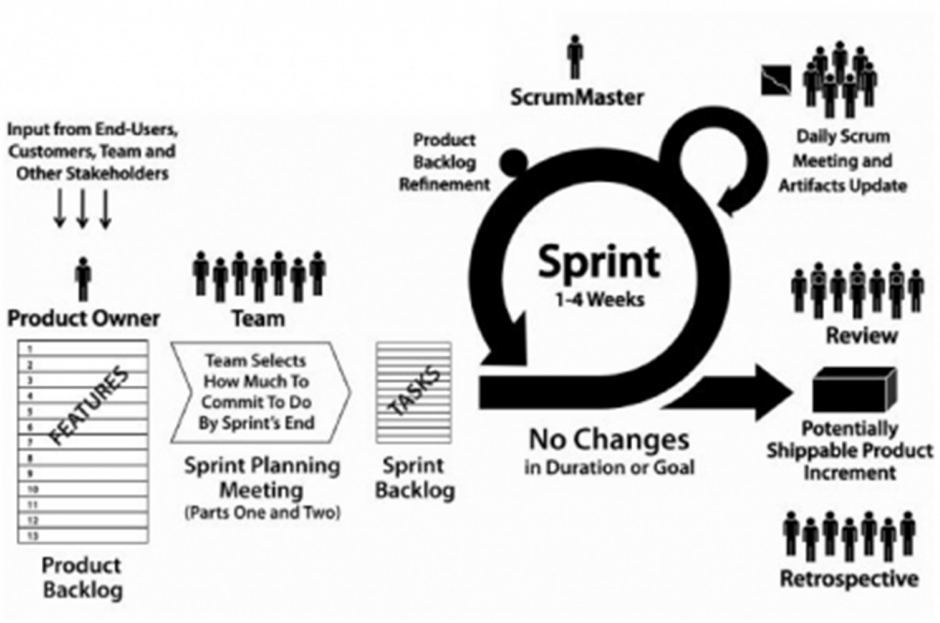
- **Inspección**

Los usuarios de Scrum deben inspeccionar con frecuencia los Artefactos y el progreso hacia los objetivos del sprint para detectar variaciones no deseadas. Sin embargo, estas inspecciones no deben realizarse con demasiada frecuencia para no obstaculizar el avance del trabajo.

- **Adaptación**

Si un inspector determina que uno o más aspectos de un proceso se desvían fuera de los límites aceptables, y que el producto resultante será inaceptable, se debe ajustar el proceso o el material que está siendo procesado. Este tipo de ajustes debe realizarse tan pronto como sea posible para minimizar aún más la desviación.

La Figura 5 muestra un diagrama general de la metodología Scrum.



[5]

Figura 5. Diagrama de metodología Scrum

3.4.1.3. Roles

Scrum define 3 roles:

- **Scrum Master**

Es el responsable del cumplimiento de las reglas de un marco de Scrum técnico, asegurando que se entienden en la organización, y se trabaja conforme a ellas. Proporciona la asesoría y formación necesaria al propietario del producto y al equipo. Realiza su trabajo con un modelo de liderazgo servil: al servicio y en ayuda del equipo y del propietario del producto.

Proporciona:

- Asesoría y formación al equipo para trabajar de forma auto-organizada y con responsabilidad de equipo.
- Revisión y validación de la pila del producto.
- Moderación de las reuniones.
- Resolución de impedimentos que en el *sprint* pueden entorpecer la ejecución de las tareas. Gestión de las "dinámicas de grupo" en el equipo.
- Configuración, diseño y mejora continua de las prácticas de Scrum en la organización. Respeto de la organización y los implicados, con las pautas de tiempos y formas de Scrum.

- **Equipo de desarrollo (Development Team)**

Lo forman el grupo de profesionales que realizan el incremento de cada sprint. Se recomienda que un equipo Scrum tenga entre 5 y 9 personas. Más allá de 9 resulta más difícil mantener la comunicación directa, y se manifiestan con más intensidad los roces habituales de la dinámica de grupos (que comienzan a aparecer típicamente a partir de 6 personas). En el cómputo del número de miembros del equipo de desarrollo no se consideran ni el Scrum Master ni el propietario del producto. No se trata de un grupo de trabajo formado por un arquitecto, diseñador o analista, programadores y testers. Es un equipo multifuncional, en el que todos los miembros trabajan de forma solidaria con responsabilidad compartida. Es posible que algunos miembros sean especialistas en áreas concretas, pero la responsabilidad es el incremento de cada sprint y recae sobre el equipo de desarrollo en conjunto. Las principales responsabilidades, más allá de la auto-organización y uso de tecnologías ágiles, son las que marcan la diferencia entre "grupo de trabajo" y "equipo". Un grupo de trabajo es un conjunto de personas que realizan un trabajo, con una asignación específica de tareas, responsabilidades y siguiendo un proceso o pautas de ejecución. Los operarios de una cadena de producción, forman un grupo de trabajo: aunque tienen un jefe común, y trabajan en la misma organización, cada uno responde por su trabajo. El equipo tiene espíritu de colaboración, y un propósito común: conseguir el mayor valor posible para la visión del cliente. Un equipo Scrum responde en su conjunto. Trabaja de forma cohesionada y auto-organizada. No hay un gestor para delimitar, asignar y coordinar las tareas. Son los propios miembros los que lo realizan. En el equipo:

- Todos conocen y comprenden la visión del propietario del producto.
- Aportan y colaboran con el propietario del producto en el desarrollo de la pila del producto.
- Comparten de forma conjunta el objetivo de cada sprint y la responsabilidad del logro.
- Todos los miembros participan en las decisiones.
- Se respetan las opiniones y aportes de todos.
- Todos conocen el modelo de trabajo con Scrum.

- **Propietario de producto (Product Owner)**

El propietario del producto es quien toma las decisiones del cliente. Su responsabilidad es el valor del producto. Para simplificar la comunicación y toma de decisiones es necesario que este rol recaiga en una única persona. Si el cliente es una organización grande, o con varios departamentos, puede adoptar la forma de comunicación interna que considere oportuna, pero en el equipo de desarrollo sólo se integra una persona en representación del cliente, y ésta debe tener el conocimiento suficiente del producto y las atribuciones necesarias para tomar las decisiones que le corresponden. En resumen, el propietario de producto es quien:

- Decide en última instancia cómo será el resultado final, y el orden en el que se van construyendo los sucesivos incrementos: qué se pone y qué se quita de la pila del producto, y cuál es la prioridad de las funcionalidades.
- Conoce el plan del producto, sus posibilidades y plan de inversión, así como del retorno esperado a la inversión realizada, y se responsabiliza sobre fechas y funcionalidades de las diferentes versiones del mismo.

En los desarrollos internos para la propia empresa, suele asumir este rol el product manager o el responsable de marketing. En desarrollos para clientes externos, el responsable del proceso de adquisición del cliente. Según las circunstancias del proyecto es posible incluso que delegue en el equipo de desarrollo, o en alguien de su confianza, pero la responsabilidad siempre es suya. Para ejercer este rol es necesario:

- Conocer perfectamente el entorno de negocio del cliente, las necesidades y el objetivo que se persigue con el sistema que se está construyendo.
- Tener la visión del producto, así como las necesidades concretas del proyecto, para poder priorizar eficientemente el trabajo.
- Disponer de atribuciones y conocimiento del plan del producto suficiente para tomar las decisiones necesarias durante el proyecto, incluidas para cubrir las expectativas previstas de retorno de la Inversión del proyecto.
- Recibir y analizar de forma continua retroinformación del entorno de negocio (evolución del mercado, competencia, alternativas) y del proyecto (sugerencias del equipo, alternativas técnicas, pruebas y evaluación de cada incremento).

Es además recomendable que el propietario de producto:

- Conozca Scrum para realizar con solvencia las tareas que le corresponden:
  - Desarrollo y administración de la pila del producto.
  - Exposición de la visión e historias de usuario [Anexo 2], y participación en la reunión de planificación de cada sprint.
- Conozca y haya trabajado previamente con el mismo equipo.

La organización debe respetar sus decisiones y no modificar prioridades ni elementos de la pila del producto.

3.4.1.4. Artefactos

Los artefactos son las herramientas que los practicantes de Scrum usan para hacer los procesos visibles.

- **Pila del producto (*Product Backlog*)**

La pila del producto es el inventario de funcionalidades, mejoras, tecnología y corrección de errores que deben incorporarse al producto a través de los sucesivos *sprints*. Representa todo aquello que esperan el cliente, los usuarios, y en general los interesados. Todo lo que suponga un trabajo que debe realizar el equipo debe estar reflejado en esta pila. Estos son algunos ejemplos de posibles entradas a una pila de producto:

- Permitir a los usuarios la consulta de las obras publicadas por un determinado autor.
- Reducir el tiempo de instalación del programa.
- Mejorar la escalabilidad del sistema.
- Permitir la consulta de una obra a través de un API web.

La pila de requisitos del producto nunca se da por completada, está en continuo crecimiento y evolución. Al comenzar el proyecto incluye los requisitos inicialmente conocidos y mejor entendidos, y conforme avanza el desarrollo, y evoluciona el entorno en el que será usado, se va desarrollando. En definitiva su continuo dinamismo refleja aquello que el producto necesita incorporar para ser el más adecuado a las circunstancias, en todo momento. Para comenzar el desarrollo se necesita la visión del objetivo de negocio que se quiere conseguir con el proyecto, comprendida y conocida por todo el equipo, y elementos suficientes en la pila para llevar a cabo el primer sprint. Habitualmente se comienza a elaborar la pila con el resultado de una reunión de "tormenta de ideas" donde colabora todo el equipo partiendo de la visión del propietario del producto. El formato de la visión no es relevante. Según los casos, puede ser una presentación informal del responsable del producto, un informe de requisitos del departamento de marketing, u otros. Sin embargo, sí es importante disponer de una visión real, comprendida y compartida por todo el equipo. El propietario del producto mantiene la pila ordenada por la prioridad de los elementos, siendo los más prioritarios los que confieren mayor valor al producto, o por alguna razón resultan más necesarios, y determinan las actividades de desarrollo inmediatas.

El detalle de los requisitos en la pila del producto debe ser proporcional a la prioridad: Los elementos de mayor prioridad deben tener mayor nivel de comprensión y detalle que los del resto. De esta forma el equipo de desarrollo puede descomponer un elemento de prioridad alta en tareas con la precisión suficiente para ser hecho en un sprint. Los elementos de la pila del producto que pueden ser incorporados a un sprint se denominan "preparados" o "accionables" y son los que pueden seleccionarse en la reunión de planificación del sprint.

- **Pila del Sprint (*Sprint Backlog*)**

La pila del sprint es la lista que descompone las funcionalidades de la pila del producto (historias de usuario) en las tareas necesarias para construir un incremento: una parte completa y operativa del producto. La realiza el equipo durante la reunión de planificación del sprint, auto-

asignando cada tarea a un miembro del equipo, e indicando en la misma lista cuánto tiempo o esfuerzo se prevé que falta para terminarla. La pila del sprint descompone el trabajo en unidades de tamaño adecuado para monitorizar el avance a diario, e identificar riesgos y problemas sin necesidad de procesos de gestión complejos. Es también una herramienta para la comunicación visual directa del equipo.

Condiciones:

- Realizada de forma conjunta por todos los miembros del equipo.
- Cubre todas las tareas identificadas por el equipo para conseguir el objetivo del sprint.
- Sólo el equipo la puede modificar durante el sprint.
- Las tareas demasiado grandes deben descomponerse en otras más pequeñas. Se deben considerar "grandes" las tareas que necesitan más de un día para realizarse.
- Es visible para todo el equipo. Idealmente en un tablero o pared en el mismo espacio físico donde trabaja el equipo.

Formato y soporte

Son soportes habituales:

- Tablero físico o pared.
- Hoja de cálculo.
- Herramienta colaborativa o de gestión de proyectos.

Y sobre el más adecuado a las características del proyecto, oficina y equipo, lo apropiado es diseñar el formato más cómodo para todos, teniendo en cuenta los siguientes criterios:

- Incluir la siguiente información: Pila del sprint, persona responsable de cada tarea, estado en el que se encuentra y tiempo de trabajo que queda para completarla.
- Incluir sólo la información estrictamente necesaria.
- Debe servir de medio para registrar en cada reunión diaria del sprint, el tiempo que le queda a cada tarea.
- Facilitar la consulta y la comunicación diaria y directa del equipo.

Durante el sprint, el equipo actualiza a diario en ella los tiempos pendientes de cada tarea. Al mismo tiempo, con estos datos traza el gráfico de avance o trabajo consumido (burn-down).

➢ Incremento

El incremento es la parte de producto producida en un *sprint*, y tiene como característica el estar completamente terminada y operativa, en condiciones de ser entregada al cliente. No se deben considerar como Incremento a prototipos, módulos o sub-módulos, ni partes pendientes de pruebas o integración. Idealmente en Scrum:

- Cada elemento de la pila del producto se refiere a funcionalidades entregables, no a trabajos internos del tipo "diseño de la base de datos".
- Se produce un "incremento" en cada iteración.

Sin embargo es una excepción frecuente el primer sprint. En el que objetivos del tipo "contrastar la plataforma y el diseño" pueden resultar necesarios, e implican trabajos de diseño o desarrollo de prototipos para contrastar las expectativas de la plataforma o tecnología que se va a emplear. Teniendo en cuenta esta excepción habitual, si el proyecto o el sistema requiere documentación, o procesos de validación y verificación documentados, éstos también tienen que estar realizados para considerar que el incremento está "hecho".

➢ Burn Charts (gráficos)

Los "*Burn Charts*" muestran la relación entre tiempo y alcance. El tiempo se posiciona en el eje horizontal de las X y el alcance en el eje vertical de las Y. Estos gráficos enseñan cuanto trabajo el equipo ha completado en un periodo determinado de tiempo. Cada vez que algo nuevo se termina la línea del gráfico se mueve hacia arriba.

Un "*Burn Down chart*" denota cuanto trabajo aún falta por completar. Generalmente, se espera que el trabajo restante vaya disminuyendo en el tiempo en base al trabajo que se completa. Algunas veces el trabajo pendiente puede sufrir de cambios repentinos cuando se agregan o remueven cosas en el alcance. Estos casos se denotan con saltos notables en los gráficos: hacia arriba o hacia abajo según se agregue o se quite trabajo respectivamente.

La Figura 6 es un ejemplo de *Sprint Burn Down Chart*, donde se ve la disminución de las horas de las tareas para el sprint conforme se avanza en el tiempo y se completan las historias de usuario.

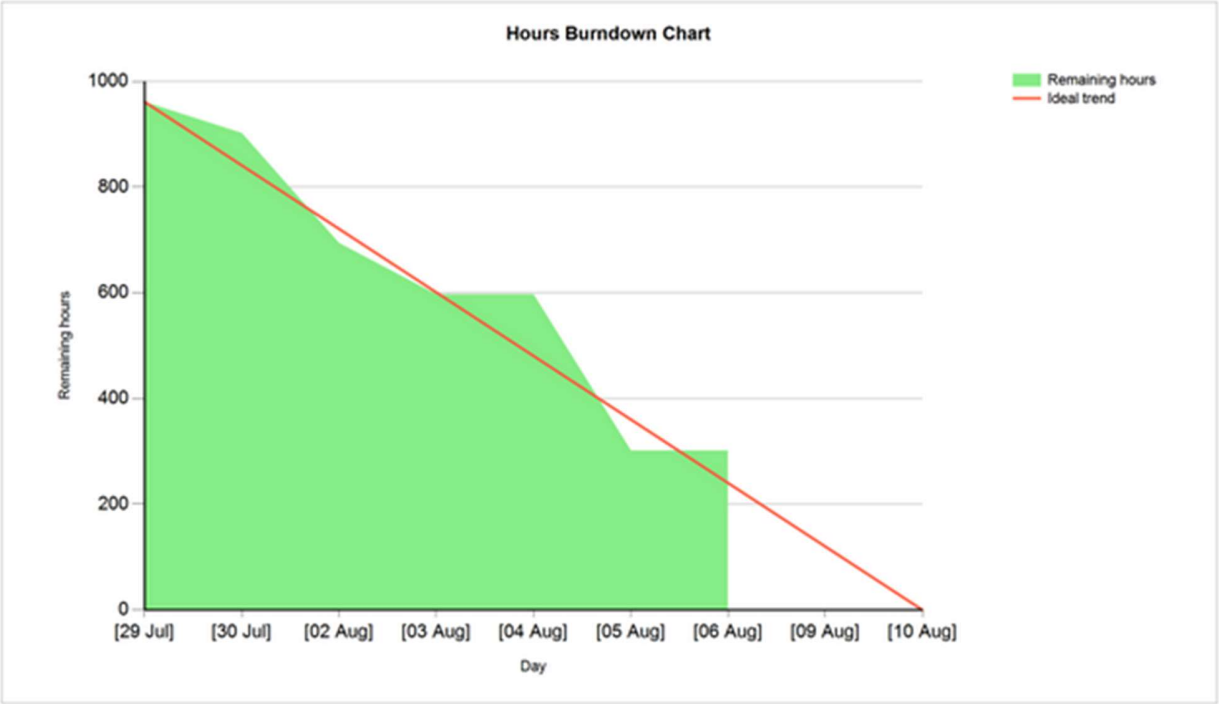


Figura 6. Ejemplo Burndown Chart

- **Task Board (pizarra de tareas)**

Aquí es donde se visualiza el trabajo para el sprint actual. Cuando las tareas de un equipo están visibles para todos, existe un riesgo mucho menor de que alguna actividad importante se olvide. La pizarra más simple consiste de 3 columnas: To Do (por hacer), In Progress (en progreso) y Done (completados).

Las tareas que se mueven por la pizarra ayudan a aumentar la visibilidad, ayudando también a inspeccionar la situación actual y adaptarse según se necesite. El Task Board también ayuda a los stakeholders (otras partes interesadas) a ver el progreso que el equipo ha hecho.

La Figura 7 es un ejemplo de un Task Board simple:

[6]

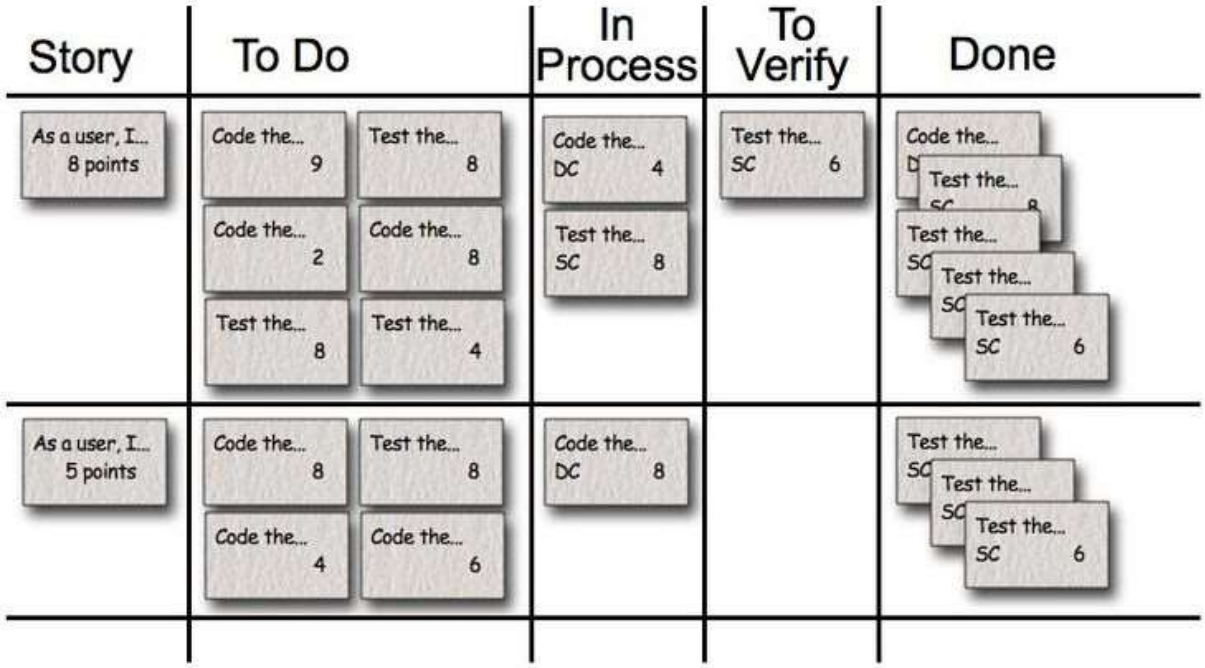


Figura 7. Scrum Task Board genérico

3.4.1.5. Eventos

Scrum identifica cuatro eventos formales para la Inspección y la Adaptación: Planificación de Sprint, Scrum Diario, Revisión del Sprint y Retrospectiva del Sprint.

- **Sprint Planning**

En esta reunión se toman como base las prioridades y necesidades de negocio del cliente, y se determinan cuáles y cómo van a ser las funcionalidades que se incorporarán al producto en el siguiente sprint. Se trata de una reunión conducida por el responsable del funcionamiento del marco Scrum a la que deben asistir el propietario del producto y el equipo completo, y a la que también pueden asistir otros implicados en el proyecto. La reunión puede durar una jornada de trabajo completa, cuando se trata de planificar un sprint largo (de un mes de duración) o un tiempo proporcional para planificar un sprint más breve. Esta reunión debe dar respuesta a dos cuestiones:

- ¿Qué se entregará al terminar el sprint?
- ¿Cuál es el trabajo necesario para realizar el incremento previsto, y cómo lo llevará a cabo el equipo?

➢ **Daily Meeting**

Reunión diaria breve, de no más de 15 minutos, en la que el equipo sincroniza el trabajo y establece el plan para las 24 horas siguientes.

Se recomienda realizarla de pie junto a un tablero con la pila del sprint y el gráfico de avance del sprint, para que todos puedan compartir la información y anotar. En la reunión está presente todo el equipo, y pueden asistir también otras personas relacionadas con el proyecto o la organización, aunque éstas no pueden intervenir.

En esta reunión cada miembro del equipo de desarrollo explica:

- Lo que ha logrado desde el anterior Scrum diario.
- Lo que va a hacer hasta el próximo Scrum diario.
- Si están teniendo algún problema, o si prevé que puede encontrar algún impedimento.

Y actualiza sobre la pila del sprint el esfuerzo que estima pendiente en las tareas que tiene asignadas, o marca como finalizadas las ya completadas.

Al final de la reunión:

- El equipo refresca el gráfico de avance del sprint, con las estimaciones actualizadas.
- El Scrum Master realiza las gestiones adecuadas para resolver las necesidades o impedimentos identificados.

El equipo es el responsable de esta reunión, no el Scrum Master y no se trata de una reunión de "inspección" o "control" sino de comunicación entre el equipo para compartir el estado del trabajo, chequear el ritmo de avance y colaborar en posibles dificultades o impedimentos.

➢ **Sprint Review**

Reunión realizada al final del sprint para comprobar el incremento. No debe durar más de 4 horas, en el caso de revisar *sprints* largos. Para *sprints* de una o dos semanas, con una o dos horas de duración debería ser suficiente.

Objetivos:

- El propietario del producto comprueba el progreso del sistema. Esta reunión marca, a intervalos regulares, el ritmo de construcción, y la trayectoria que va tomando la visión del producto.
- El propietario del producto identifica las funcionalidades que se pueden considerar "hechas" y las que no.
- Al ver y probar el incremento, el propietario del producto, y el equipo en general obtienen *feedback* relevante para revisar la pila del producto.
- Otros ingenieros y programadores de la empresa también pueden asistir para conocer cómo trabaja la tecnología empleada.

- **Sprint Retrospective**

Reunión que se realiza tras la revisión de cada sprint, y antes de la reunión de planificación del siguiente, con una duración recomendada de una a tres horas, según la duración del sprint terminado. En ella el equipo realiza autoanálisis de su sobre su forma de trabajar, e identifica fortalezas y puntos débiles. El objetivo es consolidar y afianzar las primeras, y planificar acciones de mejora sobre los segundos. El hecho de que se realice normalmente al final de cada sprint lleva a veces a considerarlas erróneamente como reuniones de "revisión de sprint", cuando es aconsejable tratarlas por separado, porque sus objetivos son diferentes. El objetivo de la revisión del sprint es analizar "QUÉ" se está construyendo, mientras que una reunión retrospectiva se centra en "CÓMO" lo estamos construyendo, con el objetivo de analizar problemas y aspectos mejorables. Las reuniones "retrospectivas" realizadas de forma periódica por el equipo para mejorar la forma de trabajo, se consideran cada vez más un componente del marco técnico de Scrum, si bien no es una reunión para seguimiento de la evolución del producto, sino para mejora del marco de trabajo.

3.4.2. eXtreme Programming

Extreme Programming (XP) es una metodología de desarrollo de ingeniería software, cuyos principios se basan en identificar los cambios y tratarlos; poniendo énfasis en prever posibles cambios y adaptarse a ellos si existiesen (Figura 8). Es un tipo de metodología ágil ya que considera que adaptar los cambios de requisitos sobre la marcha en cualquier punto de la vida del proyecto refleja una foto más fiel de la realidad y por tanto expresa una mejor aproximación.

XP define los requisitos al comienzo y posteriormente invierte el esfuerzo de manera dinámica en controlar posibles cambios que se produzcan a lo largo de la vida del mismo.

Siendo XP una disciplina para el desarrollo de software basada en los valores de simplicidad, comunicación, feedback, coraje y respeto. Aúna la adopción de las mejores metodologías de desarrollo software para la consecución de un proyecto dado y lo aplica dinámicamente durante el ciclo de vida del proyecto reuniendo al equipo completo junto a prácticas simples, con la realimentación suficiente para permitirle al equipo ver en dónde está y ajustar las prácticas a su situación única.

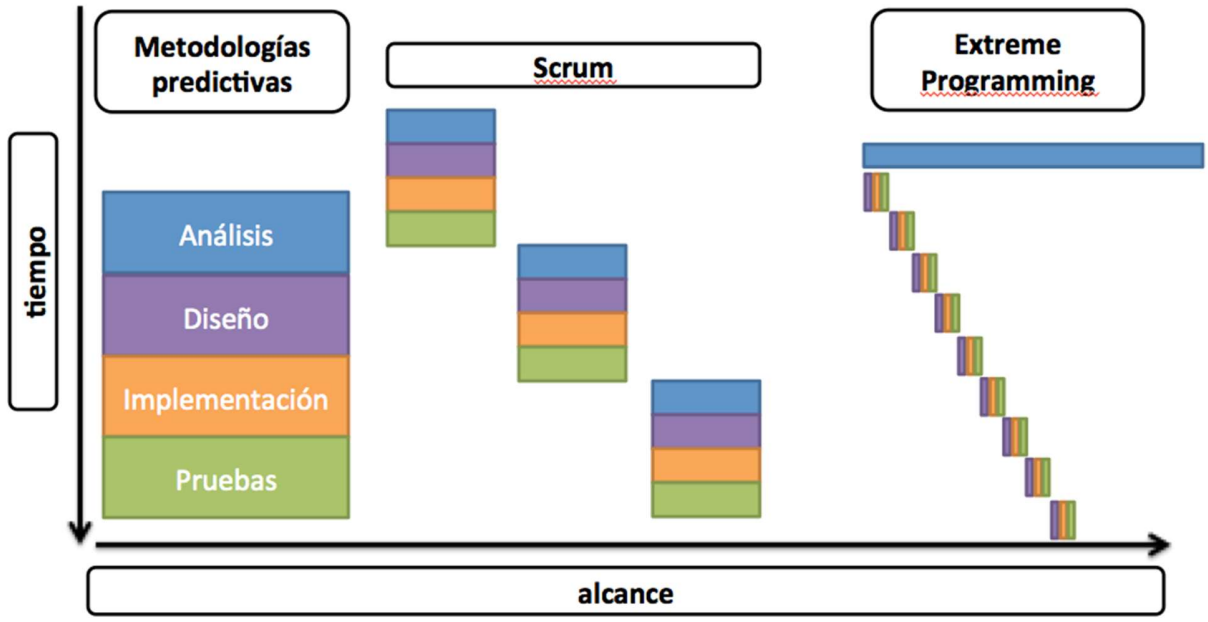


Figura 8. Fases del ciclo de vida del desarrollo en XP

Los proyectos XP se definen con cuatro variables coste, tiempo, calidad y alcance. De estas cuatro variables solo tres pueden ser establecidas externamente al grupo de desarrolladores de trabajo, clientes y jefes de proyecto. Los proyectos se realizan en ciclos cortos que llamamos iteraciones y típicamente los proyectos suelen tener entre 10 a 15 iteraciones.

Los proyectos XP se dividen además en cuatro fases exploración, planificación, de iteraciones y puesta en producción.

- **Fase de Exploración**

En la fase de exploración se define el alcance general del proyecto. El cliente identifica la necesidad y los desarrolladores estiman los tiempos de desarrollo en base a la información que se les brinda. Las estimaciones de tiempo deben ser claras y concisas, se basaran en los datos de alto nivel aportados y pueden estar sujetas a variaciones cuando se hagan análisis en mayor detalle en las iteraciones. Esta fase tiene una duración aproximada de dos semanas generalmente.

- **Fase de planificación**

En la fase de planificación las unidades organizativas externas (clientes y gerentes) y las unidades organizativas internas (desarrolladores) acuerdan el orden en que se deberían implementar las etapas para cubrir las necesidades expuestas. Esta fase suele ser de corta duración. Se establecen puntos de reunión y el resultado a esta fase es un plan de entregas o "Release Plan", donde se detalla las reglas y prácticas a tener en cuenta para la consecución del proyecto.

- **Fase de iteraciones**

Esta fase es el "eje" principal en el ciclo de vida del desarrollo de proyectos XP. Las funcionalidades de esta fase son desarrolladas y generan en cada iteración un entregable funcional que identifica la entrega implementada. Al principio de cada iteración se establece una serie de tareas necesarias de análisis, ya que en la fase de planificación no se establece un análisis tan detallado de cada uno de los entregables. Se analiza con el cliente todos los datos que son necesarios para su posterior desarrollo e implementación, generando un entregable. Por tanto, el cliente debe participar activamente en esta fase. Las iteraciones es una manera de medir el avance del proyecto, con lo que una iteración entregada sin errores es una medida clara de avance.

- **Fase de puesta en producción**

La fase de puesta en producción se da siempre que se finaliza una iteración sin errores. Además el cliente puede desear que se vaya poniendo en producción cada iteración o que se ponga en producción al completar todas las iteraciones. En esta fase no se realizan más desarrollos funcionales pero sí que se pueden dar desarrollos que necesiten de realizar tareas de ajuste "fine tunning".

Como hemos identificado en la fase de planificación en esta fase se presenta un plan de entregas, donde también quedan identificados las buenas prácticas y reglas que se van a considerar durante la consecución del proyecto. Algunos de los **conceptos y prácticas más importantes de Extreme Programming** son:

- **Equipo completo**

Todos los contribuyentes de un proyecto XP se sientan juntos, son miembros de un mismo equipo. El equipo tiene que incluir a un representante del negocio, el "Cliente", quien provee requerimientos, establece prioridades, y guía al proyecto. Lo ideal es que el Cliente o uno de sus asistentes sea un usuario final que conoce el entorno y lo que se necesita. Por supuesto, el equipo incluye a los programadores. El equipo puede incluir testers, que ayudan al Cliente a definir las pruebas de aceptación. Los analistas pueden servir como asistentes del Cliente, ayudándolo a definir los requerimientos. Suele haber un coach, que ayuda al equipo a mantener el rumbo y facilitar el proceso. Puede haber un manager, que brinda recursos, se encarga de la comunicación externa y coordina actividades. Ninguno de estos roles es propiedad exclusiva de una persona: todos en un equipo XP contribuyen de la manera que pueden. Los mejores equipos no tienen especialistas, sino contribuyentes generales con habilidades especiales.

- **Planificación**

La planificación en XP responde dos preguntas clave del desarrollo de software: predecir qué se habrá terminado para la fecha de entrega, y determinar qué hacer después. Se hace énfasis en guiar al proyecto en vez de predecir exactamente lo que se necesitará y cuánto tiempo tomará hacerlo. Hay dos pasos claves en la planificación de XP

- Planificación de la Entrega, es una práctica en donde el Cliente presenta las características deseadas a los programadores, y los programadores estiman la dificultad. Teniendo las estimaciones de coste, y sabiendo la importancia de las características, el Cliente establece un plan para el proyecto. Los planes iniciales de entregas son necesariamente imprecisos: ni las prioridades ni las estimaciones son sólidas, y tampoco sabremos lo rápido que trabaja el equipo hasta que empiece a trabajar. Sin embargo, incluso el primer plan de entrega es lo suficientemente preciso como para tomar decisiones, y el equipo XP revisa de forma regular el plan.
- Planificación de la Iteración, es la práctica en donde el equipo establece el rumbo cada dos de semanas. Los equipos XP construyen software en iteraciones de dos semanas, y entregan software útil al finalizar cada iteración. Durante la Planificación de la Iteración, el Cliente presenta las características deseadas para las siguientes dos semanas. Los programadores las descomponen en tareas, y estiman su coste (a un nivel de detalle más fino que durante la Planificación de la Entrega). El equipo entonces se compromete a terminar ciertas características basándose en la cantidad de trabajo que pudieron terminar en la iteración anterior.

Estos pasos de planificación son muy simples y le brindan al cliente muy buena información y excelente flexibilidad para guiar al proyecto. Cada dos semanas se hace completamente visible el progreso. No existe el "90% terminado" en XP: una historia está terminada, o no lo está. Este foco en la transparencia resulta en una bonita paradoja: por un lado, con tanta visibilidad el Cliente está en la posición de cancelar el proyecto si el progreso no es suficiente. Por otro lado, como el progreso es tan visible, y hay completa libertad para decidir qué se hará después, los proyectos XP tienden a entregar más de lo necesario, con menos presión y estrés.

- **Pruebas automatizadas**

Como parte de la presentación de cada característica deseada, el Cliente también define una o más pruebas de aceptación automatizadas para mostrar que la característica funciona. El equipo construye estas pruebas y las utiliza para demostrar al cliente y a ellos mismos que la característica se implementó de forma correcta. La automatización es importante porque, por la presión del tiempo, se suelen saltar las pruebas manuales.

- **Pequeñas entregas**

Los equipos XP realizan entregas pequeñas de dos formas importantes:

1. Primero, el equipo entrega software probado y funcionando, que aporta el valor de negocio elegido por el Cliente en cada iteración. El Cliente puede usar este software para cualquier propósito, sea como evaluación o incluso liberarlo para los usuarios finales. El aspecto más importante es que el software sea visible y entregado al cliente, al final de cada iteración. Esto hace que todo se mantenga abierto y tangible.
2. Segundo, los equipos XP también entregan software de forma frecuente a los usuarios finales. Los proyectos web XP entregan a diario, los proyectos internos de forma mensual.

- **Diseño simple**

Los equipos de XP construyen software sobre un diseño simple. Empiezan simple, y a través de las pruebas de programación y las mejoras al diseño, lo mantienen así. Cualquier equipo XP mantiene el diseño para que sea el justo y necesario para cumplir la funcionalidad actual del sistema. No hay desperdicio, y el software siempre está listo para lo que sigue.

El diseño en XP no es algo que se realiza en una "única vez", o que se hace completo al principio, sino que es algo que se hace todo el tiempo. Hay diseño durante la planificación de la entrega y la planificación de la iteración; además, los equipos hacen sesiones rápidas de diseño y revisiones de re-ingeniería durante todo el proyecto. En un proceso iterativo e incremental resulta esencial tener un buen diseño, por esto se hace tanto énfasis en él durante toda la vida del proyecto.

- **Programación por parejas**

En XP todo el software productivo se escribe en pareja, dos programadores sentados frente a un mismo ordenador. Esta práctica asegura que todo el código productivo fue revisado por al menos otro programador, y genera mejores diseños, mejores pruebas y mejor código.

Puede parecer ineficiente que dos programadores hagan el "trabajo de un programador", pero lo contrario es cierto. Los estudios sobre la programación por parejas muestran que las parejas producen mejor código en aproximadamente el mismo tiempo que un programador trabajando solo.

Muchos programadores son reacios la programación por parejas antes de probarla. Se necesita tiempo y práctica para hacerlo bien, y es necesario aplicarlo por varias semanas para ver los resultados. El 90% de los programadores que aprenden a trabajar de esta forma lo prefieren, por lo que la programación por parejas es una práctica recomendada para todos los equipos.

Además de generar mejor código y pruebas, la programación por parejas también sirve para comunicar el conocimiento a través de los equipos. Como las parejas cambian, todos obtienen los beneficios del conocimiento especializado de las personas. Los programadores aprenden, mejoran sus habilidades, se vuelven más valiosos para el equipo y para la organización.

- **TDD**

Extreme Programming pone mucho énfasis el feedback, y en el desarrollo de software el buen feedback necesita de buenas pruebas. Los mejores equipos de XP practican Desarrollo Guiado por Pruebas (TDD), trabajando en ciclos muy cortos entre la adición y ejecución de cada nueva prueba. Casi sin esfuerzo, los equipos producen código con casi un 100% de cobertura de pruebas, lo cual es un gran avance para la mayoría de los proyectos.

No es suficiente con escribir pruebas, debemos ejecutarlas. Aquí también XP es extremo. Estas "pruebas del programador" o "pruebas unitarias" se juntan, y cada vez que cualquier programador sube código al repositorio (y las parejas lo suelen hacer dos o más veces al día), cada una de las pruebas unitarias debe ejecutarse con éxito. Esto significa que los programadores tienen un feedback inmediato sobre cómo están avanzando.

- **Refactorización**

Extreme Programming se enfoca en entregar valor de negocio en cada iteración. Para lograr esto a lo largo de todo el proyecto, el software debe estar bien diseñado. La alternativa es retrasarse hasta detenerse por completo. Es por esto que XP utiliza un proceso de mejora continua del diseño llamado *Refactoring*, sacado del libro de Fowler “Refactoring: Improving the Design of Existing Code “ [7].

El proceso de refactorización se enfoca en eliminar la duplicidad (una clara señal de diseño pobre), y en incrementar la cohesión del código, disminuyendo al acoplamiento. Hace ya treinta años que se considera a la alta cohesión y al bajo acoplamiento como el máximo logro de un buen diseño. El resultado es que los equipos XP comienzan con un diseño simple y bueno, y siempre tienen un diseño simple y bueno para el software. Esto les permite mantener la velocidad, y en general suele acelerar la velocidad a medida que el proyecto avanza.

La refactorización se apoya en pruebas completas que aseguran que, a medida que el diseño evoluciona, nada se rompa. Las pruebas del cliente y unitarias son críticas para permitir este proceso. Las prácticas de XP se apoyan entre sí: son más poderosas que por separado.

- **Integración continua**

Los equipos XP realizan integraciones de código muchas veces por día. La integración infrecuente de código lleva a serios problemas en un proyecto de software. En el momento de la integración suelen aparecen problemas que no se detectaron en ninguna de las pruebas unitarias del software. Si la integración se realiza con grandes bloques de código se producen largos períodos de "congelación" del código. La congelación de código (o *"code freeze")* significa que por largos períodos los programadores no pueden agregar nuevas características, aunque sea necesario. Esto debilita nuestra posición en el mercado y con los usuarios finales.

- **Propiedad colectiva de código**

En un proyecto XP, cualquier pareja de programadores puede mejorar cualquier porción de código en cualquier momento. Esto significa que el código se beneficia de la atención de las personas, lo cual resulta en una mayor calidad de código y en menos defectos. También hay otro beneficio importante: cuando los individuos son dueños del código, se suelen agregar características en lugares equivocados cuando un programador descubre que necesita agregar algo en un lugar que "no es suyo". El dueño está muy ocupado para hacerlo, por lo que le programador escribe la característica en su propio código, en donde no corresponde. Esto lleva a código difícil de mantener, lleno de duplicación y con baja (mala) cohesión.

La propiedad colectiva de código sería un problema si las personas trabajaran a ciegas en código que no entienden. XP evita estos problemas a través de dos técnicas: las pruebas del programador (unitarias) atrapan los errores, y la programación de a pares significa que la mejor forma de trabajar con código poco familiar es estar en pareja con un experto. Además de asegurar buenas modificaciones cuando se las necesita, esta práctica ayuda a compartir el conocimiento en todo el equipo.

- **Estándares de código**

Los equipos XP usan un estándar de código en común, de manera que el código del sistema se vea como si fuera escrito por una única persona muy competente. No importa mucho el

estándar en sí mismo: lo importante es que el código se vea familiar, para permitir la propiedad colectiva.

- **Metáfora del sistema**

Los equipos XP desarrollan una visión común sobre cómo funciona el programa, que llamamos la "metáfora". La metáfora es una descripción evocativa simple sobre cómo funciona el programa; por ejemplo "este programa funciona como una colmena de abejas, que salen a buscar polen y lo traen de vuelta a la colmena", podría ser una descripción para un sistema de recuperación de información a través de agentes.

A veces no aparece una metáfora poética. En ese caso, con o sin una imágen viva, los equipos XP usan un sistema común de nombres para asegurarse que todos entiendan cómo funciona el sistema, en dónde buscar la funcionalidad que queremos, o cómo encontrar el lugar adecuado para agregar algo nuevo.

3.4.3. Crystal

Recordando lo visto en apartados anteriores definimos un proceso como ágil cuando el desarrollo de software es incremental, con entregas pequeñas de software funcional en ciclos rápidos y cortos; es cooperativo, el cliente y los desarrolladores trabajan juntos constantemente y mantienen una comunicación dinámica y cercana. Además podemos modificarlo de manera sencilla, ya que al ser fácil de aprender, modificar y estar documentado es adaptable y por lo tanto permite realizar cambios sin importar el hito o el momento del proceso en el que nos encontremos. Por ello, decimos que las metodologías ágiles tienen la capacidad de ser muy adaptables a los cambios en el sistema deseado por el cliente, y que se orientan más a las personas que al proceso de desarrollo.

No hay una única metodología Crystal, ya que esta se basa en que proyectos diferentes requieren de tipos de diferentes de metodologías, y debe ser cada equipo de desarrolladores quienes identifiquen a partir de unas características comunes, considerar la metodología que mejor aplica, ajustándola a la situación y necesidad concreta a resolver. Por lo tanto estas metodologías a diferencia de otras vistas en este documento están pensadas para una mayor variedad de tipos de proyectos y organizaciones, destacando sobre todo en empresas grandes.

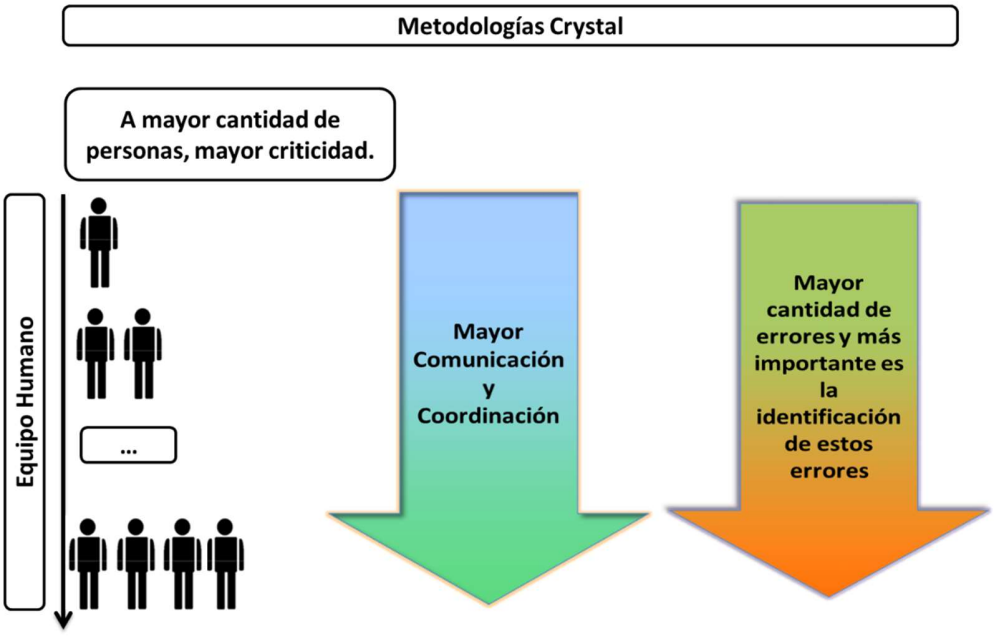


Figura 9. Implicaciones del tamaño del equipo en la metodología Crystal

El nombre de las metodologías Crystal viene de que cada proyecto software puede caracterizarse según dos dimensiones, tamaño y criticidad (Figura 9), donde estas dos variables representan por un lado las personas involucradas en el proyecto y por otro las consecuencias de los errores que pueda producir el software desarrollado. Las metodologías Crystal, las asociamos mediante colores en proyectos grandes según aspectos como son los de necesitar una mayor coordinación y comunicación en el proyecto, o la de identificar un fallo que pueda causar mayores problemas. De esta manera identificamos la familia de metodologías Crystal como:

- Clear, para equipos de hasta 8 personas o menos.
- Amarillo, de entre 10 y 20 personas.
- Naranja, para equipos entre 20 y 50 personas.
- Roja, entre 50 y 100 personas.

A más personas en el proyecto, más coordinación. A más criticidad en el software, más rigurosidad en el proceso. El factor más determinante en cualquier caso, es la comunicación entre los participantes en el proyecto. La otra gran clave de metodologías Crystal, común a casi todas las metodologías ágiles, es que lo más determinante para el éxito, o fracaso, de un proyecto son las personas y el trabajo dinámico y en equipo; siendo esta una de las claves que determinan el éxito (o fracaso) de un proyecto software.

Las metodologías de Crystal se categorizan a partir del número de personas involucradas y la criticidad que se genera. Todas las metodologías Crystal cumplen con 7 propiedades esenciales que las definen (Figura 10).

Figura 10. Propiedades de metodología Crystal

A continuación identificamos estas siete propiedades:

- **Entregas frecuentes**, en base a un ciclo de vida iterativo e incremental.

  En función del proyecto puede haber desde entregas semanales hasta trimestrales. En Scrum las entregas son, máximo, cada 4 semanas, en las metodologías Crystal se contemplan muchas más opciones.

- **Mejora continua**.

  Las iteraciones ayudan a ir ajustando el proyecto, a ir mejorándolo.

- **Comunicación cara a cara**.

- **Expresión y participación activa.**

  Todo el mundo puede expresar su opinión sin miedos, teniéndosele en cuenta, considerándose su opinión, etc.

- **Enfoque del entorno**.

  Períodos sin interrupción de 2h con el equipo. Los objetivos y prioridades deben estar claros, definiendo así tareas concretas. Rendimiento del desarrollo de software a partir del entorno físico.

- **Fácil acceso a usuarios expertos.**

  Las metodologías Crystal (a diferencia de otras como XP) no exigen que los usuarios estén continuamente junto al equipo de proyecto (no todas las organizaciones pueden hacerlo), sí que, como mínimo, semanalmente debe haber reuniones y los usuarios deben estar accesibles.

- **Integración continua.**

  Entorno técnico con pruebas automatizadas, gestión de la configuración e integración continua.

3.4.4. Kanban

La metodología Kanban es otra de las metodologías que forman parte de las metodologías ágiles. Su objetivo es la de gestionar la consecución general de las tareas. La palabra Kanban viene del japonés que significa "tarjetas visuales". Kanban permite generar resultados altamente efectivos y eficientes en sistemas productivos, ya que ha contribuido a generar un panorama óptimo y competitivo.

El origen de Kanban se encuentra en los sistemas de producción "Just In Time" (JIT), donde es necesaria el uso de tarjetas para identificar las necesidades dentro de la cadena de producción, pero actualmente también es usado en procesos de desarrollo de software. Las principales ventajas de esta metodología es que es fácil de usar, actualizar y asumir por parte del equipo, ya que trata de gestionar las tareas de forma muy visual, lo que permite ver el estado de los proyectos de manera rápida, así como también establecer las pautas de desarrollo de tareas de manera más efectiva. Un ejemplo de panel Kanban es el de la Figura 11.

[8]

Figura 11. Ejemplo panel Kanban

Como cualquiera de las metodologías anteriores Kanban también tiene una serie de principios que la diferencian de otras metodologías ágiles y que identificamos a continuación:

3.4.4.1. Principios de Kanban

- **Calidad garantizada**

No existe margen de error en Kanban ya que todo debe salir a la primera. Por esto no es rápido.

- **Aumento de la eficiencia**

Kanban se basa en hacer solamente lo justo y necesario.

- **Mejora continua**

Kanban no solamente es un método de gestión sino que también es un sistema de mejora de desarrollo de proyectos, según objetivos que se desean alcanzar.

- **Flexibilidad**

Las tareas se entrantes se priorizan según las necesidades del momento. Es capaz de dar respuesta a tareas imprevistas de manera eficiente.

Usar Kanban implica crear un tablón de tareas que deben desarrollarse de manera constante o a un ritmo sostenible. Hace viable la realización de entregas en cualquier momento según la prioridad y estructura integral que tenga el proyecto a realizar. Aunque estas sean cualidades que resaltan la metodología Kanban sobre otras metodologías ágiles, enfatizo que no existen como tal metodologías mejores sino que es dependiendo del proyecto, de la naturaleza de la organización y como están conformados sus procesos internos, lo que hace elegir una metodología frente a otras. Kanban es en este sentido una metodología que se ajusta mejor en organizaciones que requieren flexibilidad en la ejecución de tareas, seguimiento de estas, priorización, supervisión del equipo de trabajo y los informes entregados. Las principales reglas de Kanban por tanto son la de:

1. Visualizar el trabajo y fases del ciclo de producción o flujo de trabajo.
2. Determinar la duración del trabajo en curso (WIP – “Work in progress”)
3. Medir el tiempo que se tarda en completar una tarea (Lead time)

Para usar esta metodología y sus reglas por tanto debemos tener claros los siguientes aspectos:

3.4.4.2. Aspectos y reglas para usar Kanban

- **Definir flujo de trabajo del proyecto.**

Consiste en definir el tablón de trabajo, que deberá estar accesible y ser visible para los componentes del equipo. Cada columna del tablón corresponderá a un flujo de tareas, que nos

servirá para saber en qué situación se encuentra cada proyecto. Hay tantas columnas como estados por los que pasa cada una tarea desde su inicio hasta su finalización. El tablón es continuo a diferencia de Scrum y no hay fases del ciclo de producción establecidas sino que se definen según el caso en cuestión o el modelo general que se aplicara a cualquier proyecto dentro de una organización. Por lo tanto, el objetivo de esta visualización (tablón) es que quede claro el trabajo a realizar, en qué está trabajando cada persona y tener clara la prioridad de las tareas. La metodología Kanban no define roles, ya que tener un papel asignado y tareas asociadas a dicho papel hacen que el individuo adopte una única dimensión del proyecto. Si se le asigna un nuevo puesto de trabajo con un nuevo rol, el individuo debe de adaptarse y que no haya una resistencia al cambio. Por eso Kanban trata de evitar esa resistencia emocional, no definiendo los roles, ya entiende que la ausencia de papeles es una ventaja para el equipo.

- **Definir fases del ciclo del proyecto.**

Kanban se basa en el desarrollo incremental y divide el trabajo en distintas partes, no de las tareas, sino de diferentes puntos que agilizan el proceso de producción. Las fases del ciclo de producción o flujo de trabajo se deben decidir según el caso, no hay nada acotado. Cada parte se escribe en una etiqueta que luego se coloca en la fase en la que se encuentre dependiendo del estado en el que se encuentre dicha parte. Además de identificar la parte a la que nos estamos refiriendo en la etiqueta también indicamos una serie de observaciones básicas que la referencian, por ejemplo las interdependencias que hay entre partes o tareas a la hora de ejecutar el desarrollo de las mismas. Las etiquetas intentan clarificar al máximo el trabajo a realizar, la prioridad y duración que deben tener, haciendo una descripción visual al proceso y las tareas a desarrollar.

- **No olvidar lema de Kanban "Stop Starting, start finishing"**

Con este lema lo que se intenta priorizar es el finalizar las tareas que están en curso antes de comenzar nuevas tareas. Las tareas en curso deben tener duración limitada, además de que existe un número limitado de tareas en cada fase. En la práctica lo que sucede o se intenta establecer es que no se puede abrir una tarea sin finalizar otra. Para ello se debe determinar el límite del trabajo en curso (*Work In Progress*). Donde una de las principales ideas del Kanban es que el trabajo en curso debería estar limitado, es decir, que el número de tareas que se pueden realizar en cada fase debe ser algo conocido. Independientemente de si un proyecto es grande o pequeño, simple o complejo, hay una cantidad de trabajo óptima que se puede realizar sin dejar de ser eficientes, por ejemplo, puede ser que realizar diez tareas a la vez nos lleve una semana, pero hacer dos cosas a la vez nos lleve sólo unas horas, lo que nos permite hacer quince tareas en la semana. Por lo tanto en Kanban se debe definir cuantas tareas como máximo puede realizarse en cada fase del ciclo de trabajo simultáneamente.

- **Control del flujo**

La metodología Kanban no solo se aplica a un único proyecto, sino que se pueden mezclar tareas y proyectos diferentes. El flujo de trabajo debe ser relativamente constante y se ponen a la cola las tareas priorizando las tareas más importantes. Además se intenta interrumpir lo menos posible

al empleado, realizando un seguimiento pasivo. Se debe medir el tiempo en completar una tarea, a este tiempo que se le llama *Lead time*. El *Lead time* cuenta desde que se hace una petición hasta que se hace la entrega. Otro indicador importante que no debemos olvidar es el *cycle time*. El *cycle time* mide desde que el trabajo sobre una tarea comienza hasta que termina. Por lo tanto con el *lead time* se mide lo que ven los clientes y lo que esperan, mientras que con el *cycle time* se mide más el rendimiento del proceso. Estos dos indicadores son los más importantes y necesarios para el control y la mejora continua.

3.4.5. Scrumban

La metodología Scrumban es la agrupación de las mejores prácticas de la metodología ágil Scrum y las mejores prácticas de la metodología ágil Kanban para la gestión de proyectos (Figura 12). Donde la premisa de Scrumban es emplear lo mejor de ambas metodologías, usándose en proyectos que al combinarlos sirvan para mejorar la productividad de un plan de empresa. Aparentemente parecidas, ambas tienen diferencias en la manera de ejecutar los proyectos tal y como hemos visto en los apartados anteriores donde las hemos comentado. Scrumban implementa elementos de ambas que se pueden complementar. Desarrolla una transición suave de Scrum a Kanban. Para ello existen varios niveles de transición. Siendo estas transiciones secuencias evolutivas a través de prácticas eficientes.

[9]

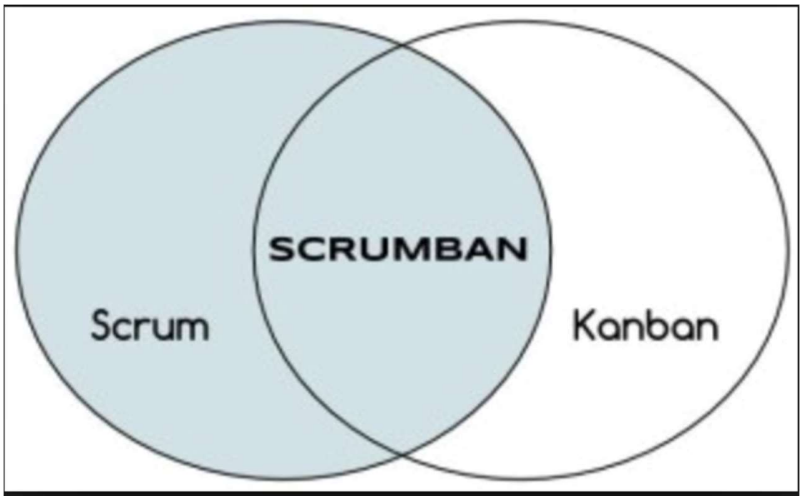


Figura 12. Intersección Scrum y Kanban

El ciclo de trabajo es el mismo que Kanban ya que tiene etapas relacionadas entre sí, además de tener reuniones diarias cortas con el equipo de trabajo. Al igual que sus predecesores consta de un panel compuesto por parte Scrum y por parte Kanban. Donde en el panel de Scrum metemos las historias priorizadas y ordenadas verticalmente; cada columna traza generalmente el flujo de tareas, mientras que el panel de Kanban contiene directamente las tareas sin estimar. Las historias se deben subdividir en tareas que se estiman y planifican en reuniones, marcándolas por colores. Mientras que las tareas sin estimar se cuantifican en horas al final y se les asigna un grupo:

1. Deja lo que estés haciendo y ponte con esta tarea
2. Cuando termines la tarea que estés haciendo te pones hacer esta la primera
3. Deberías hacer esta tarea pero si no compromete la entrega del sprint.

Scrumban se usa en muchas ocasiones en gestionar las tareas previstas con el método Scrum y planificar los errores con el método Kanban. Las etapas en Scrumban, donde se categorizan las tareas, no solo son las de Kanban (sin comenzar, en progreso, finalizadas) sino que tras ser revisadas mediante Scrum, se añaden más categorías (probadas, entregadas).

Los equipos en Scrumban deben estar organizados en pequeñas iteraciones y monitorizados con la ayuda del panel. Se deben planificar reuniones que permitan determinar que tareas deben ejecutarse en la siguiente iteración. Las tareas son añadidas al panel. Para que las iteraciones no se alarguen mucho en el tiempo las tareas tienen un tiempo de duración tal y como se vio en Kanban. En Scrumban tampoco se definen roles, el equipo mantiene los roles que tienen asignados en su organización.

Hay proyectos que se implementan mejor con este método, ya que se trata de proyectos con un alto nivel de complejidad. Scrumban se adecúa bien en proyectos de mantenimiento, en proyectos de requisitos que varíen con frecuencia y en proyectos en los que surjan errores de ejecución. En los proyectos de mantenimiento se suele necesitar de presentaciones parciales del desarrollo y por eso es interesante usar Scrumban. En los proyectos cuyos requisitos varíen constantemente, al no tener fijadas las condiciones y necesidades a cubrir, estas se van introduciendo a partir de las diferentes etapas. En los proyectos donde surjan errores en la ejecución, debemos evaluar y analizar el método usado y evaluar las tareas.

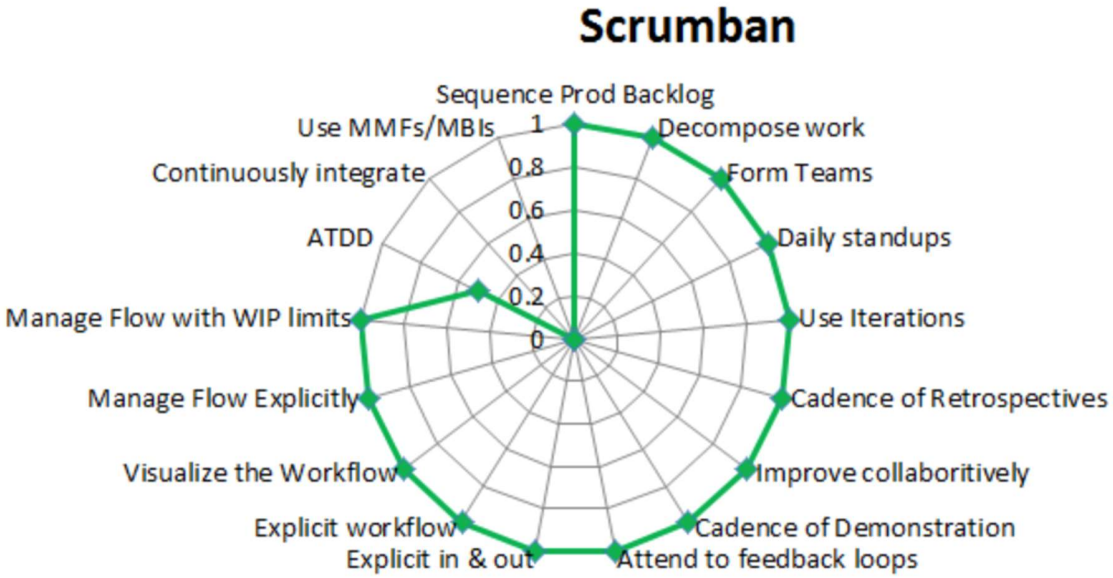


Figura 13. Diagrama características Scrumban (www.netobjectives.com)

Para usar esta metodología y sus reglas por tanto debemos tener claros los siguientes aspectos:

3.4.5.1. Aspectos y reglas para usar Scrumban

- **Iteraciones. Definir flujo de trabajo del proyecto.**

Las iteraciones como hemos identificado anteriormente deben ser cortas, garantizando la adaptación del equipo ante cambios rápidos del entorno. La duración de una iteración debe calcularse en función del número de tareas que contenga.

- **Planificar bajo demanda las tareas**

La manera de planificar en Scrumban es bajo demanda. Siempre que haya tareas que queden por hacer deben planificarse para ser implementadas. El número de tareas que deben ser implementadas para evaluar lo realizado depende del proyecto y sus características. La velocidad con la que se implementan las tareas depende del equipo desarrollador. Antes de finalizar las tareas en curso se evalúa que tareas se deben implementar en la siguiente iteración y se añaden a la categoría de tareas pendientes por hacer.

- **Priorizar las tareas**

Se debe priorizar las tareas en un orden y deben quedar identificadas la prioridad de cada tarea en el panel. Se suele poner las tareas más prioritarias en la parte de arriba de la columna y las menos prioritarias en la parte de abajo.

- **Planificación del proyecto**

Una manera de priorizar y planificar las tareas en proyectos de larga duración es mediante cubos que representan los estados de la planificación por ejemplo: (1 año, 6 meses, 3 meses). De esta manera queda reflejada claramente los requisitos del proyecto tanto a corto como a largo plazo. Los requisitos pasan de un cubo a otro dependiendo del estado de implementación que se puede desarrollar, de esta manera el equipo puede dibujar su ciclo de tareas a desarrollar bajo demanda.

- **Diseño del panel**

El panel que se construye es de primeras igual que el que teníamos en Kanban, y es cuando se evalúa y analiza las iteraciones cuando se añaden categorías que permiten visualizar la evolución del equipo.

- **Planificar dimensiones de indicadores**

Hay dos indicadores que son muy importantes en Scrumban uno de ellos es la duración de las tareas en curso (WIP- Work in Progress) y el otro es la duración del conjunto de tareas que falta por desarrollarse.

Mediante el WIP podemos evaluar la eficiencia del equipo de trabajo. Los diferentes hitos de desarrollo por los que pasan. Además, esta medida debe visualizarse en el panel en la categoría de en progreso. La duración de las tareas en curso depende del número de desarrolladores que existen en el equipo.

Medir la duración del conjunto de tareas que faltan por desarrollarse, permite ser más

productivo en la planificación de reuniones, el desarrollo de tareas y el número de tareas que pueden realizarse en paralelo.

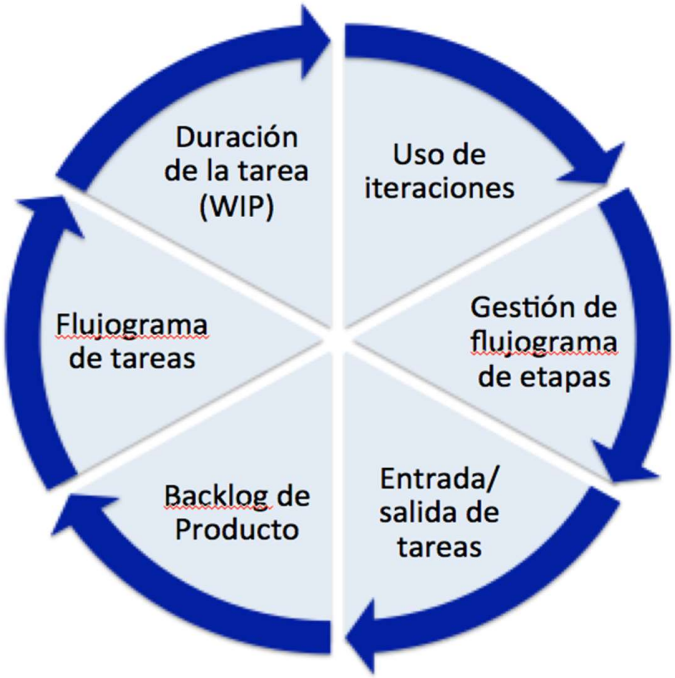


Figura 14. Flujo Scrumban

3.4.5.2. Beneficios de la metodología Scrumban

- Visualizamos el estado real del proceso de ejecución del proyecto.
- Propone soluciones a errores eventuales.
- Una evaluación de tareas desarrolladas más detallada.
- Reuniones habituales. Mayor interacción de personas del equipo.
- Mayor productividad en proyectos complejos.
- Favorece la adaptación de los componentes del equipo.

Capítulo 4

Comparación de las metodologías ágiles

El objetivo de este apartado es plantear un sistema que sirva para establecer qué metodología ágil, de las descritas en este documento, se adapta más a las necesidades de una organización o proyecto concreto. No se pretende juzgar qué metodología es mejor o peor, sino realizar una comparativa de las diferentes herramientas.

Para la selección e implantación de una metodología existe una importante labor de documentación previa y, a partir de ahí, escoger alguna de las metodologías vistas para aplicar en el día a día de nuestro trabajo.

Se ha elaborado un cuestionario que consta de dos partes, una inicial para conocer la orientación de la organización: ágil o tradicional y una segunda parte que permite conocer la metodología ágil que mejor se adapta al marco de trabajo de una organización.

4.1. Orientación de la organización

En esta fase se extraerá el enfoque de la organización, bien un enfoque tradicional o un enfoque ágil. Si la empresa sigue en un porcentaje alto de implantación de las directrices de las metodologías ágiles, se podría pasar a la segunda parte del formulario, mientras que si en esta primera fase se detecta que la cultura de trabajo es más cercana a una metodología tradicional, sería necesario conocer y adquirir las prácticas de una metodología ágil.

4.1.1. Primer formulario: Orientación tradicional vs orientación ágil

Para obtener este dato de forma objetiva, se analizará cada valor ágil y su relación con la organización.

Se han desglosado los valores del manifiesto ágil y se han dividido entre orientación ágil vs orientación tradicional, estos valores serán evaluados por la organización según una escala de importancia.

Valores de importancia:

0: Ninguna.

1: Baja importancia.

2: Media importancia.

3: Alta importancia.

ORIENTACIÓN ÁGIL		ORIENTACIÓN TRADICIONAL	
VALOR	**IMPORTANCIA**	**VALOR**	**IMPORTANCIA**
Individuo y las interacciones del equipo	x1	El proceso y las herramientas	y1
Desarrollar software que funciona	x2	Conseguir una buena documentación	y2

Colaboración con el cliente	x3	Negociación contractual	y3
Respuesta al cambio	x4	Seguimiento de un plan	y4

Tabla 1. Orientación tradicional vs orientación ágil

Siendo x1, x2, x3 y x4 los valores asignados a cada valor con enfoque ágil, e y1, y2, y3 e y4 los valores asignados a cada valor con enfoque tradicional.

❖ Caso Práctico

La siguiente tabla ha sido rellenada por un desarrollador de la organización obteniéndose los siguientes resultados. Las cabeceras descriptivas de la orientación fueron ocultadas para evitar condicionar los resultados.

ORIENTACIÓN ÁGIL		ORIENTACIÓN TRADICIONAL	
VALOR	**IMPORTANCIA**	**VALOR**	**IMPORTANCIA**
Individuo y las interacciones del equipo	3	El proceso y las herramientas	2
Desarrollar software que funciona	3	Conseguir una buena documentación	2
Colaboración con el cliente	2	Negociación contractual	2
Respuesta al cambio	3	Seguimiento de un plan	2

Tabla 2. Resultado orientación tradicional vs orientación ágil

En este caso, se demuestra que se sobrevalora lo indicado por los valores del manifiesto ágil.

4.1.2. Segundo Formulario: Cumplimiento principios ágiles

Se ha extraído a un formulario cada principio ágil y extraerá su relación con la organización.

Valores en prioridad:
0: Ninguna.
1: Baja prioridad.
2: Media prioridad.

3: Alta prioridad.

El siguiente formulario se debe entregar a miembros de la organización para que lo rellenen de acuerdo a los valores indicados previamente:

	Principios del Manifiesto Ágil	**Prioridad**
1	La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.	
2	Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.	
3	Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.	
4	La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.	
5	Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo	
6	El diálogo cara a cara es el método más efectivo para comunicar información dentro de un equipo de desarrollo.	
7	El software que funciona es la medida principal de progreso.	
8	Los procesos ágiles promueven un desarrollo sostenible. Los promotores, los desarrolladores y usuarios deberían ser capaces de mantener una paz constante.	
9	La atención continua a la calidad técnica y al buen diseño mejora la agilidad.	
10	La simplicidad es esencial.	
11	Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.	
12	En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.	
	Total	**Σ prioridades asignadas**

Tabla 3. Cumplimiento principios ágiles

Siendo los principios ágiles doce y la prioridad más alta tiene un valor de 3, entonces el valor más alto en cuanto al cumplimiento de estos principios de 36.

Según el resultado obtenido Σ *prioridades asignadas*, se deduce que las metas según el enfoque de la gerencia de sistemas de la empresa se orientan en un $\frac{\sum prioridades\ asignadas}{36} * 100\%$ al cumplimiento de los principios ágiles.

❖ Caso práctico

Indicadores de los principios ágiles en una organización según un desarrollador de la organización:

	Principios del Manifiesto Ágil	**Prioridad**
1	La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.	2
2	Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.	2
3	Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.	2
4	La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.	2
5	Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo	3
6	El diálogo cara a cara es el método más efectivo para comunicar información dentro de un equipo de desarrollo.	3
7	El software que funciona es la medida principal de progreso.	3
8	Los procesos ágiles promueven un desarrollo sostenible. Los promotores, los desarrolladores y usuarios deberían ser capaces de mantener una paz constante.	3
9	La atención continua a la calidad técnica y al buen diseño mejora la agilidad.	3
10	La simplicidad es esencial.	3
11	Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.	2

12	En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.	2
	Total	**30**

Tabla 4. Ejemplo cumplimiento principios ágiles

Siendo los principios ágiles 12 y la prioridad más alta tiene un valor de 3, entonces el valor más alto en cuanto al cumplimiento de estos principios de 36.

Según el resultado obtenido de 30, se deduce que las metas a seguir de la organización según el desarrollador se orientan en un 83% al cumplimiento íntegro o total de los principios ágiles.

El resultado será más representativo cuantos más miembros de la empresa realicen el cuestionario.

4.2. Elección de una metodología ágil

En este paso se evalúa la forma de trabajo de la empresa basándose en los cuatro puntos de vista de Iacovelli [10]. Para ello, se ha elaborado un nuevo formulario agrupando estos cuatro puntos: Uso, capacidad de agilidad, aplicación, procesos y productos (Figura 15). Cada uno de ellos con sus respectivos atributos, cuyos valores serán asignados por la empresa en evaluación.

4.2.1. Modelo de Iacovelli

El objetivo del estudio realizado por Iacovelli, "Framework para la clasificación de metodologías ágiles", es clasificar los métodos a través de cuatro puntos de vista, cada uno representando un aspecto de las metodologías. Cada punto de vista se caracteriza por un conjunto de atributos.

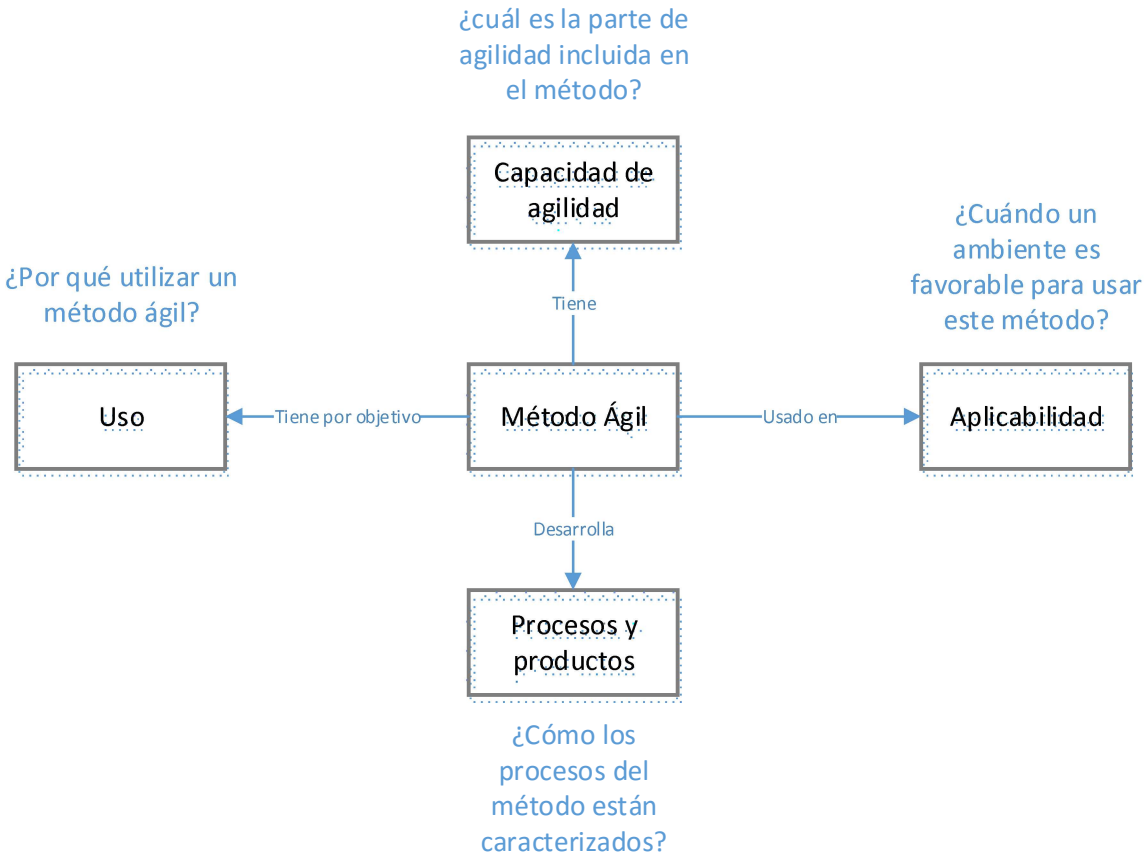


Figura 15. Cuatro vistas de las metodologías ágiles (Iacovelli)

❖ USO

Refleja por qué utilizar metodologías ágiles. Los atributos de esta vista tratan de evaluar todos los beneficios de que el equipo de desarrollo y el cliente obtienen utilizando este tipo de metodologías: incremento de la productividad, calidad y satisfacción.

Las metodologías ágiles integran los cambios en el proceso de desarrollo, aportan reglas y directrices para trabajar en proyectos con requisitos cambiantes manteniendo fechas de entrega. Las metodologías ágiles aportan flexibilidad.

Los atributos de este punto de vista son:

- Adaptarse a los entornos turbulentos.
- Satisfacción del usuario final.
- Favorable a la descentralización (outsourcing internacional).
- Aumento de la productividad.

- Cumplimiento de un nivel de calidad.
- El respeto de las fechas de entrega.
- Cumplimiento de los requisitos.

❖ CAPACIDAD DE AGILIDAD

Representa cuál es la parte ágil de la metodología. Los atributos de esta vista representan todos los aspectos del concepto de agilidad y su evaluación refleja que aspectos están incluidos en una metodología.

Una metodología de desarrollo de software está compuesta por un ciclo de vida. En Ingeniería del Software existen diferentes ciclos de vida, como por ejemplo modelo en V, modelo en espiral, etc. De acuerdo a la cronología de aparición de las metodologías ágiles, la mayoría de las metodologías derivan directamente del modelo en espiral. Esto se explica ya que las dos principales características de modelo en espiral son un ciclo de vida iterativo e incremental. Por tanto, los cambios de requisitos se pueden ir integrando en cada iteración, de manera que el plan de trabajo no tiene que ser modificado, irá cambiando a lo largo de las iteraciones. Otro punto interesante es la duración de las iteraciones, con iteraciones cortas aumenta el número de reuniones con el cliente para definir y detallar sus necesidades de forma incremental.

En cuanto a la interacción de las personas, las metodologías ágiles tienden a romper las relaciones contractuales entre los clientes y los equipos de desarrollo. Esta relación se expresa en este punto de vista por el atributo de colaboración. Un equipo ágil es un tipo de organización holográfica en la que cada miembro tiene el conocimiento del sistema en su conjunto, así que, si un miembro deja el equipo, no se ha perdido conocimiento.

El principal concepto de la agilidad son los procesos ligeros. Generalmente, las metodologías ágiles incluyen menos documentación. Las pruebas son una práctica muy importante, así como la refactorización.

Los atributos de este punto de vista son:

- Indicadores de cambio.
- Colaboración.
- Los requisitos funcionales pueden cambiar.
- Los recursos humanos pueden cambiar.
- Integración de los cambios.
- Nivel de intercambio de conocimientos (baja, alta).
- De peso ligero.

- Requisito no funcional puede cambiar.
- Centrado en las personas.
- Reactividad (al comienzo del proyecto, cada etapa, cada iteración).
- Política de refactorización.
- Iteraciones cortas.
- Política de pruebas.
- Plan de trabajo se puede cambiar.

❖ APLICABILIDAD

El objetivo de esta vista es mostrar el impacto de los aspectos ambientales en el método. Representa cuando el entorno es favorable para la aplicación de metodologías ágiles. Este aspecto se describe por atributos, cada uno correspondiente a una característica del entorno.

Los atributos de este punto de vista son:

- Grado de interacción entre los miembros del equipo (baja, alta).
- El grado de interacción con el cliente (baja, alta).
- Grado de interacción con los usuarios finales (baja, alta).
- Grado de integración de la novedad (baja, alta).
- La complejidad del proyecto (baja, alta).
- Los riesgos del proyecto (baja, alta).
- Tamaño del proyecto (pequeño, grande).
- La organización del equipo (auto-organización, la organización jerárquica).
- El tamaño del equipo (pequeño, grande).

❖ PROCESOS Y PRODUCTOS

La vista de los procesos y productos representa cómo se caracteriza la metodología. Los atributos caracterizarán a los procesos ágiles por dos dimensiones y listarán los productos de las actividades del proceso.

El proceso se compone de dos dimensiones. La primera dimensión son las actividades de desarrollo de software cubiertas por las metodologías ágiles. La segunda representa el nivel de abstracción de sus directrices y reglas. Estas dos dimensiones se evalúan con atributos de esta vista.

Los atributos de los procesos y los productos son:

Nivel de abstracción de las normas y directrices

- Gestión del proyecto
- Descripción de procesos
- Normas y orientaciones concretas sobre las actividades y productos

Las actividades cubiertas por el método ágil

- Puesta en marcha del proyecto
- Definición de requisitos
- Modelado
- Código
- Pruebas unitarias
- Pruebas de integración
- Prueba del sistema
- Prueba de aceptación
- Control de calidad
- Sistema de uso

Productos de las actividades del método

- Modelos de diseño
- Comentario del código fuente
- Ejecutable
- Pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de aceptación
- Informes de calidad
- Documentación de usuario

4.2.2. Tercer Formulario: Elección de una metodología ágil

A continuación se presentan cuatro formularios con las características de cada uno de los puntos de vista de Iacovelli.

❖ **USO**

Rellene el siguiente formulario con Verdadero (V) o Falso (F) en cada una de las premisas.

	Premisa	**Respuesta**
1	Respeto de las fechas de entrega	
2	Cumplimiento de los requisitos	
3	Respeto al nivel de calidad	
4	Satisfacción del usuario final	
5	Entornos turbulentos	
6	Favorable a la deslocalización	
7	Aumento de la productividad	

Tabla 5. Valoración del Uso

❖ **CAPACIDAD DE AGILIDAD**

Rellene el siguiente formulario con Verdadero (V) o Falso (F) en cada una de las premisas.

	Premisa	**Respuesta**
1	Iteraciones cortas	
2	Colaboración	
3	Centrado en las personas	
4	Política de refactorización	
5	Política de pruebas	
6	Integración de los cambios	
7	Procesos livianos	
8	Los requisitos funcionales pueden cambiar	
9	Los requisitos no funcionales pueden cambiar	
10	El plan de trabajo puede cambiar	
11	Los recursos humanos pueden cambiar	
12	Cambiar los indicadores	

13	Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	
14	Intercambio de conocimientos (BAJO, ALTO)	

Tabla 6. Valoración de la Capacidad de agilidad

❖ APLICACIÓN

Rellene el siguiente formulario con Verdadero (V) o Falso (F) en cada una de las premisas.

	Premisa	Respuesta
1	Tamaño del proyecto (PEQUEÑO, GRANDE)	
2	La complejidad del proyecto (BAJA, ALTA)	
3	Los riesgos del proyecto (BAJO, ALTO)	
4	El tamaño del equipo (PEQUEÑO, GRANDE)	
5	El grado de interacción con el cliente (BAJA, ALTA)	
6	Grado de interacción con los usuarios finales (BAJA, ALTA)	
7	Grado de interacción entre los miembros del equipo (BAJA, ALTA)	
8	Grado de integración de la novedad (BAJA, ALTA)	
9	La organización del equipo (AUTO-ORGANIZACIÓN, ORGANIZACIÓN JERÁRQUICA)	

Tabla 7. Valoración de la Aplicación

❖ PROCESOS Y PRODUCTOS

Rellene los siguientes formularios con Verdadero (V) o Falso (F) en cada una de las premisas.

Nivel de abstracción de las normas y directrices:

	Premisa	Respuesta
1	Gestión de proyectos	
2	Descripción de procesos	
3	Normas y orientaciones concretas sobre las actividades y productos	

Tabla 8. Valoración normas y directrices ágiles

Las actividades cubiertas por el método ágil:

	Premisa	Respuesta
1	Puesta en marcha del proyecto	
2	Definición de requisitos	
3	Modelado	
4	Código	
5	Pruebas unitarias	
6	Pruebas de integración	
7	Prueba del sistema	
8	Prueba de aceptación	
9	Control de calidad	
10	Sistema de uso	

Tabla 9. Valoración actividades cubiertas por el método ágil

Productos de las actividades del método:

	Premisa	Respuesta
1	Modelos de diseño	
2	Comentario del código fuente	
3	Ejecutable	
4	Pruebas unitarias	
5	Pruebas de integración	
6	Pruebas de sistema	
7	Pruebas de aceptación	
8	Informes de calidad	
9	Documentación de usuario	

Tabla 10. Valoración de los productos de las actividades de la metodología

Los datos extraídos de este formulario se compararán con clasificación de las diferentes metodologías que se muestra a continuación (Tabla 11).

			METODOLOGÍAS ÁGILES				
			ORIENTADA AL DESARROLLO DE SOFTWARE	ORIENTADA A LA GESTIÓN DE PROYECTOS			
			XP	SCRUM	KANBAN	SCRUMBAN	CRYSTAL
USO	¿Por qué utilizar un método ágil?	Respeto de las fechas de entrega	FALSO	VERDADERO	FALSO	FALSO	VERDADERO
		Cumplimiento de los requisitos	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Respeto al nivel de calidad	FALSO	FALSO	FALSO	FALSO	FALSO
		Satisfacción del usuario final	FALSO	VERDADERO	FALSO	FALSO	FALSO
		Entornos turbulentos	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
		Favorable a la deslocalización	FALSO	VERDADERO	FALSO	VERDADERO	VERDADERO
		Aumento de la productividad	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
CAPACIDAD DE AGILIDAD	¿Cuál es la parte de agilidad incluida en el método?	Iteraciones cortas	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
		Colaboración	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Centrado en las personas	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Política de refactorización	VERDADERO	FALSO	FALSO	FALSO	FALSO
		Política de pruebas	VERDADERO	VERDADERO	FALSO	VERDADERO	FALSO
		Integración de los cambios	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
		Procesos livianos	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
		Los requisitos funcionales pueden cambiar	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
		Los requisitos no funcionales pueden cambiar	FALSO	FALSO	VERDADERO	VERDADERO	FALSO
		El plan de trabajo puede cambiar	VERDADERO	FALSO	VERDADERO	VERDADERO	FALSO
		Los recursos humanos pueden cambiar	VERDADERO	FALSO	VERDADERO	VERDADERO	VERDADERO
		Indicadores de cambio	VERDADERO	FALSO	FALSO	FALSO	FALSO
		Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	ITERACIÓN	ETAPA	ITERACIÓN	ETAPA	ETAPA
		Intercambio de conocimientos (BAJO, ALTO)	ALTO	BAJO	BAJO	BAJO	ALTO
ABILI	¿Cuándo un ambiente es	Tamaño del proyecto (PEQUEÑO, GRANDE)	PEQUEÑO	GRANDE/ PEQUEÑO	PEQUEÑO	GRANDE/ PEQUEÑO	GRANDE

	favorable para usar este método?	Complejidad del proyecto (BAJA, ALTA)	BAJA	ALTA	BAJA	ALTA	ALTA
		Riesgos del proyecto (BAJO, ALTO)	BAJO	ALTO	BAJO	ALTO	ALTO
		Tamaño del equipo (PEQUEÑO, GRANDE)	PEQUEÑO	PEQUEÑO	PEQUEÑO	PEQUEÑO	PEQUEÑO
		Interacción con el cliente (BAJA, ALTA)	ALTA	ALTA	BAJO	BAJA	BAJA
		Interacción con los usuarios finales (BAJA, ALTA)	BAJO	ALTA	BAJO	BAJA	BAJO
		Interacción entre los miembros del equipo (BAJA, ALTA)	ALTA	ALTA	BAJA	ALTA	ALTA
		Grado de integración de la novedad (BAJA, ALTA)	ALTA	ALTA	BAJA	ALTA	BAJA
		Organización del equipo (AUTO-ORGANIZACIÓN, ORGANIZACIÓN JERÁRQUICA)	AUTO-ORGANIZACIÓN	AUTO-ORGANIZACIÓN	AUTO-ORGANIZACIÓN	AUTO-ORGANIZACIÓN	AUTO-ORGANIZACIÓN
PROCESOS Y PRODUCTOS	¿Cómo están caracterizados los procesos del método?	**Nivel de abstracción de las normas y directrices**					
		Gestión del proyecto	FALSO	VERDADERO	FALSO	VERDADERO	VERDADERO
		Descripción de procesos	VERDADERO	FALSO	FALSO	FALSO	VERDADERO
		Normas y orientaciones concretas sobre las actividades y productos	VERDADERO	FALSO	FALSO	FALSO	FALSO
		Las actividades cubiertas por el método ágil					
		Puesta en marcha del proyecto	FALSO	FALSO	FALSO	FALSO	FALSO
		Definición de requisitos	VERDADERO	VERDADERO	FALSO	VERDADERO	FALSO
		Modelado	VERDADERO	VERDADERO	FALSO	FALSO	VERDADERO
		Código	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas unitarias	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de integración	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Prueba del sistema	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Prueba de aceptación	FALSO	FALSO	FALSO	FALSO	FALSO
		Control de calidad	FALSO	FALSO	FALSO	FALSO	FALSO
		Sistema de uso	FALSO	FALSO	FALSO	FALSO	FALSO
		Productos de las actividades del método ágil					
		Modelos de diseño	FALSO	VERDADERO	FALSO	VERDADERO	FALSO
		Código fuente comentado	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Ejecutable	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO

		Pruebas unitarias	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de integración	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de sistema	VERDADERO	FALSO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de aceptación	FALSO	FALSO	FALSO	FALSO	FALSO
		Informes de calidad	FALSO	FALSO	FALSO	FALSO	FALSO
		Documentación de usuario	FALSO	FALSO	FALSO	FALSO	FALSO

Tabla 11. Clasificación de metodologías ágiles

Comparando los resultados obtenidos del formulario y la tabla de clasificación anterior, se identificará la metodología que mejor se adapta a la forma de trabajo de la empresa. La metodología adecuada será la que mayor número de coincidencias tenga con el cuestionario anterior.

➢ Caso práctico

Las tablas presentadas anteriormente (Tabla 5, Tabla 6, Tabla 7, Tabla 8, Tabla 9 y Tabla 10) han sido completadas por un desarrollador de la organización y comparadas con la Tabla 11. El resultado puede verse en las siguientes tablas (Tabla 12, Tabla 13, Tabla 14, Tabla 15, Tabla 16 y Tabla 17). En los casos en los que la respuesta del desarrollador coincida con el valor asociado a la metodología se sumará 1, en caso contrario 0.

❖ USO

	Premisa	**Respuesta**
1	Respeto de las fechas de entrega	V
2	Cumplimiento de los requisitos	V
3	Respeto al nivel de calidad	V
4	Satisfacción del usuario final	V
5	Entornos turbulentos	V
6	Favorable al Off shoring	F
7	Aumento de la productividad	V

Tabla 12. Ejemplo valoración del Uso

❖ CAPACIDAD DE AGILIDAD

	Premisa	**Respuesta**
1	Iteraciones cortas	V

2	Colaboración	V
3	Centrado en las personas	V
4	Refactoring político	F
5	Prueba político	V
6	Integración de los cambios	V
7	De peso ligero	V
8	Los requisitos funcionales pueden cambiar	V
9	Los requisitos no funcionales pueden cambiar	V
10	El plan de trabajo puede cambiar	V
11	Los recursos humanos pueden cambiar	V
12	Cambiar los indicadores	V
13	Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	ITERACIÓN
14	Intercambio de conocimientos (BAJO, ALTO)	BAJO

Tabla 13. Ejemplo valoración de la Capacidad de agilidad

❖ APLICACIÓN

	Premisa	**Respuesta**
1	Tamaño del proyecto (PEQUEÑO, GRANDE)	PEQUEÑO
2	La complejidad del proyecto (BAJA, ALTA)	BAJA
3	Los riesgos del proyecto (BAJO, ALTO)	BAJO
4	El tamaño del equipo (PEQUEÑO, GRANDE)	PEQUEÑO
5	El grado de interacción con el cliente (BAJA, ALTA)	ALTA
6	Grado de interacción con los usuarios finales (BAJA, ALTA)	ALTA
7	Grado de interacción entre los miembros del equipo (BAJA, ALTA)	ALTA
8	Grado de integración de la novedad (BAJA, ALTA)	ALTA
9	La organización del equipo (AUTO-ORGANIZACIÓN, ORGANIZACIÓN JERÁRQUICA)	JERÁRQUICA

Tabla 14. Ejemplo valoración de la Aplicación

❖ PROCESOS Y PRODUCTOS

Nivel de abstracción de las normas y directrices:

	Premisa	Respuesta
1	Gestión de proyectos	V
2	Descripción de procesos	V
3	Normas y orientaciones concretas sobre las actividades y productos	F

Tabla 15. Ejemplo valoración de las normas y directrices de la metodología ágil

Las actividades cubiertas por el método ágil:

	Premisa	Respuesta
1	Puesta en marcha del proyecto	V
2	Definición de requisitos	V
3	Modelado	F
4	Código	V
5	Pruebas unitarias	V
6	Pruebas de integración	V
7	Prueba del sistema	V
8	Prueba de aceptación	V
9	Control de calidad	V
10	Sistema de uso	V

Tabla 16. Ejemplo valoración de las actividades cubiertas por la metodología ágil

Productos de las actividades del método:

	Premisa	Respuesta
1	Modelos de diseño	F
2	Comentario del código fuente	V
3	Ejecutable	V
4	Pruebas unitarias	V
5	Pruebas de integración	V
6	Pruebas de sistema	V
7	Pruebas de aceptación	V
8	Informes de calidad	V
9	Documentación de usuario	V

Tabla 17. Ejemplo valoración de los productos de las actividades de la metodología ágil

El resultado de cruzar las tablas rellenadas por el desarrollador con los valores establecidos para las metodologías evaluadas es el mostrado en la Tabla 18:

		METODOLOGÍAS ÁGILES				
		ORIENTADA AL DESARROLLO DE SOFTWARE	ORIENTADA A LA GESTIÓN DE PROYECTOS			
		XP	SCRUM	KANBAN	SCRUMBAN	CRYSTAL
USO	Respeto de las fechas de entrega	0	1	0	0	1
	Cumplimiento de los requisitos	1	1	1	1	1
	Respeto al nivel de calidad	0	0	0	0	0
	Satisfacción del usuario final	0	1	0	0	0
	Entornos turbulentos	1	1	1	1	0
	Favorable a la deslocalización	1	0	1	0	0
	Aumento de la productividad	1	1	1	1	0
CAPACIDAD DE AGILIDAD	Iteraciones cortas	1	1	1	1	0
	Colaboración	1	1	1	1	1
	Centrado en las personas	1	1	1	1	1
	Política de refactorización	0	1	1	1	1
	Política de pruebas	1	1	0	1	0
	Integración de los cambios	1	1	1	1	0
	Procesos livianos	1	1	1	1	0
	Los requisitos funcionales pueden cambiar	1	1	1	1	0
	Los requisitos no funcionales pueden cambiar	0	0	1	1	0
	El plan de trabajo puede cambiar	1	0	1	1	0
	Los recursos humanos pueden cambiar	1	0	1	1	1
	Indicadores de cambio	1	0	0	0	0
	Reactividad	1	0	1	0	0
	Intercambio de conocimientos	0	1	1	1	0
APLICABILIDAD	Tamaño del proyecto	1	1	1	1	0
	Complejidad del proyecto	1	0	1	0	0
	Riesgos del proyecto	1	0	1	0	0
	Tamaño del equipo	1	1	1	1	1
	Interacción con el cliente	1	1	0	0	0

	Interacción con los usuarios finales	0	1	0	0	0
	Interacción entre los miembros del equipo	1	1	0	1	1
	Grado de integración de la novedad	1	1	0	1	0
	Organización del equipo	0	0	0	0	0
PROCESOS Y PRODUCTOS	**Nivel de abstracción de las normas y directrices**					
	Gestión del proyecto	0	1	0	1	1
	Descripción de procesos	1	0	0	0	1
	Normas y orientaciones concretas sobre las actividades y productos	0	1	1	1	1
	Las actividades cubiertas por el método ágil					
	Puesta en marcha del proyecto	0	0	0	0	0
	Definición de requisitos	1	1	0	1	0
	Modelado	0	0	1	1	0
	Código	1	1	1	1	1
	Pruebas unitarias	1	1	1	1	1
	Pruebas de integración	1	1	1	1	1
	Prueba del sistema	1	1	1	1	1
	Prueba de aceptación	0	0	0	0	0
	Control de calidad	0	0	0	0	0
	Sistema de uso	0	0	0	0	0
	Productos de las actividades del método ágil					
	Modelos de diseño	1	0	1	0	1
	Código fuente comentado	1	1	1	1	1
	Ejecutable	1	1	1	1	1
	Pruebas unitarias	1	1	1	1	1
	Pruebas de integración	1	1	1	1	1
	Pruebas de sistema	1	0	1	1	1
	Pruebas de aceptación	0	0	0	0	0
	Informes de calidad	0	0	0	0	0
	Documentación de usuario	0	0	0	0	0
	TOTAL	**34**	31	32	32	21

Tabla 18. Metodología ágil adecuada para el ejemplo

En los resultados se observa que XP, Scrum, Kanban y Scrumban han obtenido puntuaciones muy similares. Podría aplicarse cualquiera de ellas, o incluso una opción podría ser aplicar XP para la fase de desarrollo y Scrum para la gestión de los proyectos de la organización.

Capítulo 5

Caso práctico

En este capítulo se muestra un caso práctico de un proyecto gestionado mediante metodología Scrum en el que se desarrolló una serie de funcionalidades en una plataforma BMC Remedy en su versión 7.6. Remedy es una plataforma de aplicaciones de la compañía BMC cuyo motor Action Request System se desarrolla mediante un lenguaje de programación de cuarta generación, en el que el desarrollador literalmente pinta los formularios de la aplicación y les asocia el flujo y las reglas necesarias.

A continuación se muestran las historias de usuario a implementar en el proyecto. Las historias de usuario son el equivalente a los requisitos de las metodologías tradicionales (Más detalles en Anexo 1). Dichas historias de usuario muestran:

1. Un título para la historia de usuario.
2. Un identificador numérico para la historia de usuario.
3. Una descripción de la funcionalidad a implementar.
4. Los criterios de aceptación para poder determinar que el desarrollo está finalizado
5. La prioridad que tiene dicha funcionalidad dentro del proyecto.

6. Una estimación de los puntos de historia que supone.

Los 5 primeros puntos son definidos por el *Propietario del Producto*, mientras que el punto 6 es estimado por el equipo de desarrollo.

También se muestran las tareas en las que se descompone cada historia de usuario para su implementación. Está división en tareas es fruto del análisis del equipo de desarrollo.

Figura 16. Tarjeta de la historia de usuario 1

Tareas asociadas a esta historia de usuario:

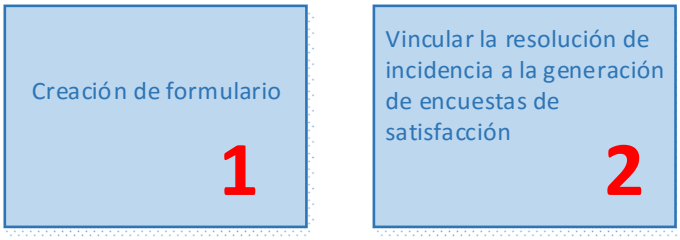


Figura 17. Tareas asociadas a la historia de usuario 1

Figura 18. Tarjeta de la historia de usuario 2

Tareas asociadas a esta historia de usuario:

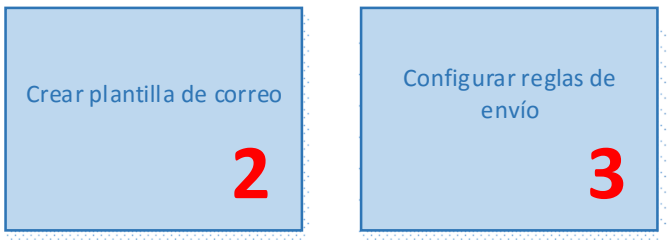


Figura 19. Tareas asociadas a la historia de usuario 2

Figura 20. Tarjeta de la historia de usuario 3

Tareas asociadas a esta historia de usuario:

Figura 21. Tareas asociadas a la historia de usuario 3

Figura 22. Tarjeta de la historia de usuario 4

Tareas asociadas a esta historia de usuario:

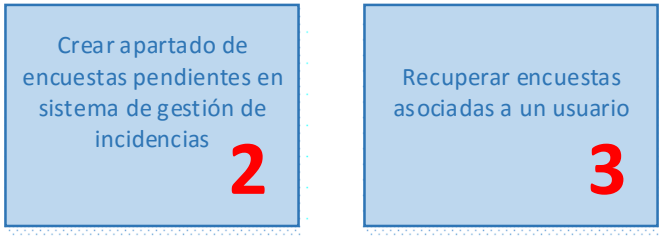


Figura 23. Tareas asociadas a la historia de usuario 4

Figura 24. Tarjeta de la historia de usuario 5

Tareas asociadas a esta historia de usuario:

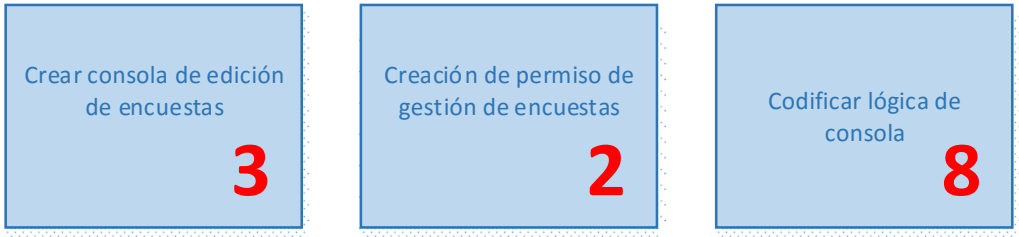


Figura 25. Tareas asociadas a la historia de usuario 5

Figura 26. Tarjeta de la historia de usuario 6

Tareas asociadas a esta historia de usuario:

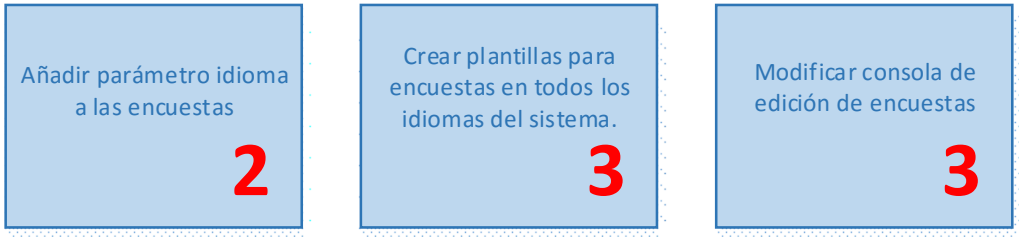


Figura 27. Tareas asociadas a la historia de usuario 6

Figura 28. Tarjeta de la historia de usuario 7

Tareas asociadas a esta historia de usuario:

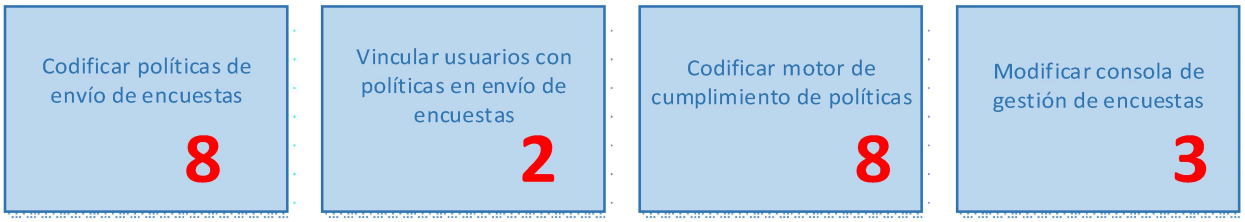


Figura 29. Tareas asociadas a la historia de usuario 7

El equipo de desarrollo está compuesto por 5 recursos que desarrollaran individualmente o por parejas según vayan considerando necesario.

A continuación se mostrará un resumen del día a día de los desarrollos realizados hasta completar todas las historias de usuario.

4.3. Sprint 0

Una vez que tenemos todas las historias de usuario, utilizamos la herramienta online gratuita Trello para montar un panel *Kanban* que nos servirá para realizar el seguimiento del proyecto.

En primer lugar, generamos una tarjeta con cada historia de usuario que tenemos que desarrollar, tal y como se observa en la Figura 30. Este listado de historias de usuario compondrá la Pila de producto (*Product Backlog)*. El equipo estimará los puntos de historia asociados a cada historia de usuario tras realizar varias votaciones en las que los miembros mostrarán su estimación de forma simultánea. Después de cada votación los miembros con menor y mayor estimación justificarán el por qué se sus estimaciones y se volverá a votar hasta alcanzar un acuerdo sobre la estimación final. Para una mayor comprensión de esta técnica de estimación consultar el punto Anexo 2 de este documento.

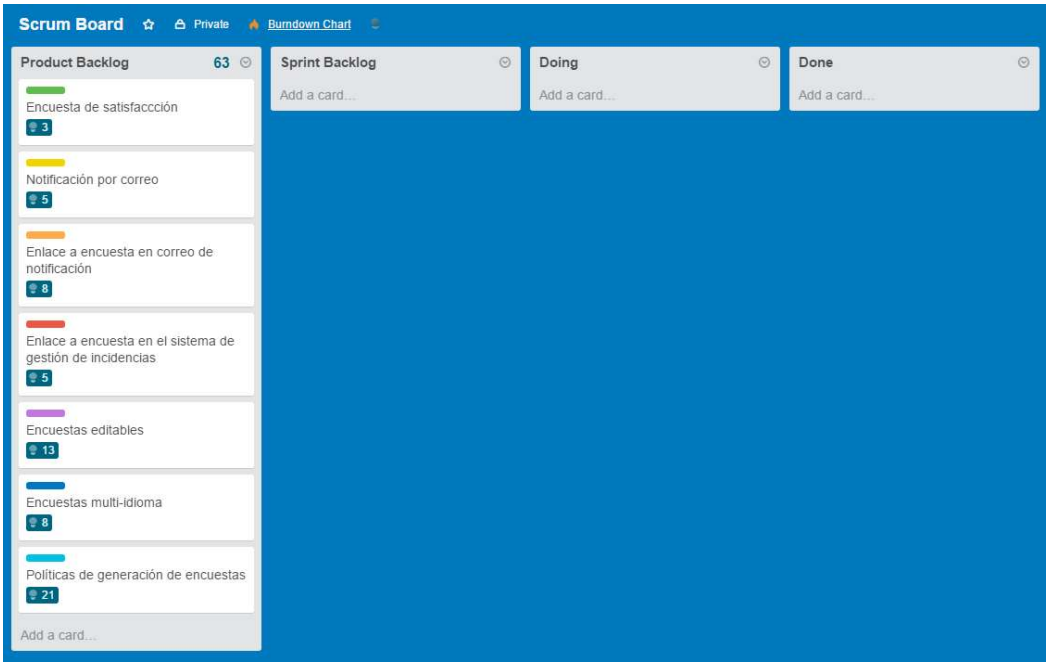


Figura 30. Product Backlog del caso práctico

A continuación ordenamos por prioridad según el criterio del Propietario del Producto (*Product Owner*). Esta ordenación nos servirá para saber en qué orden debemos comenzar a desarrollar (Figura 31).

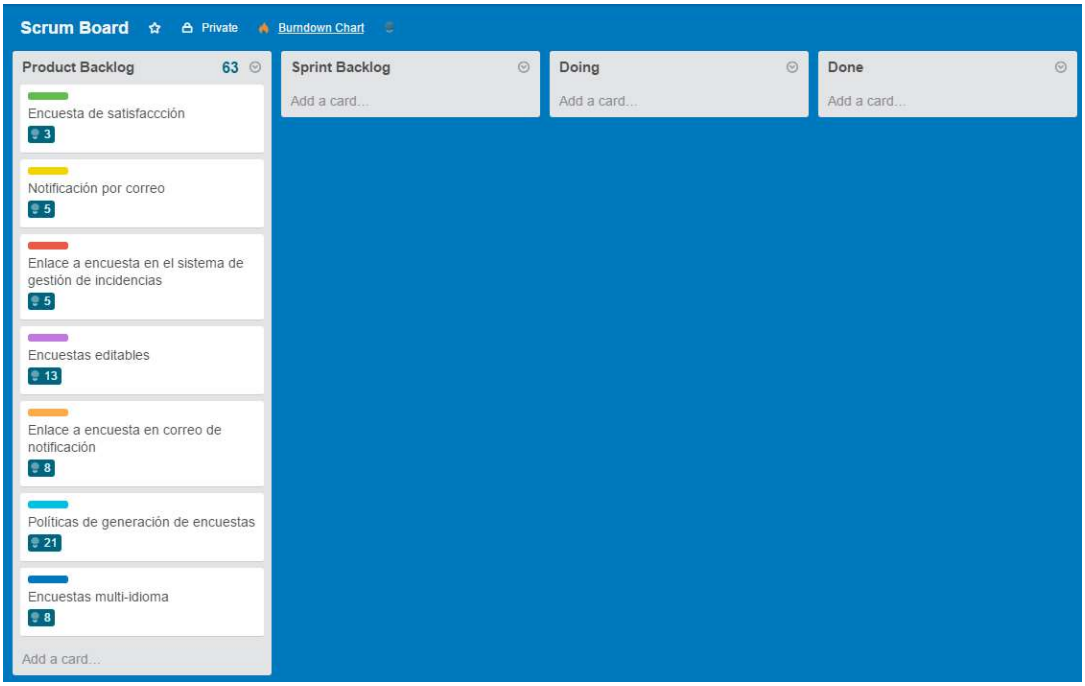


Figura 31. Product Backlog ordenado según prioridad

Una vez ordenado, el equipo selecciona las historias de usuario a desarrollar en el *sprint 1 (*Figura 32*)*.

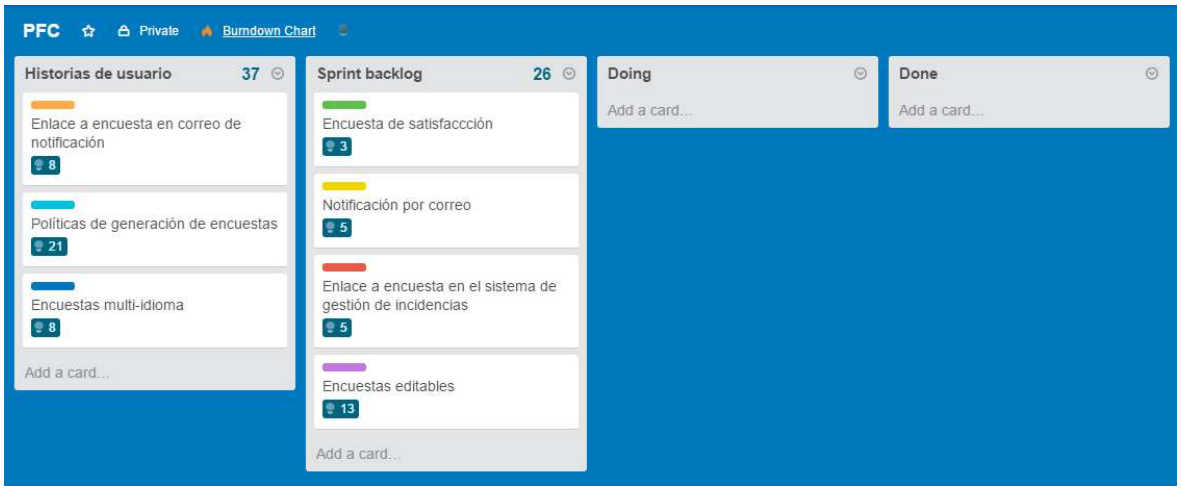


Figura 32. Selección de historias de usuario del sprint 1

El equipo descompone las historias de usuario en tareas (Figura 33).

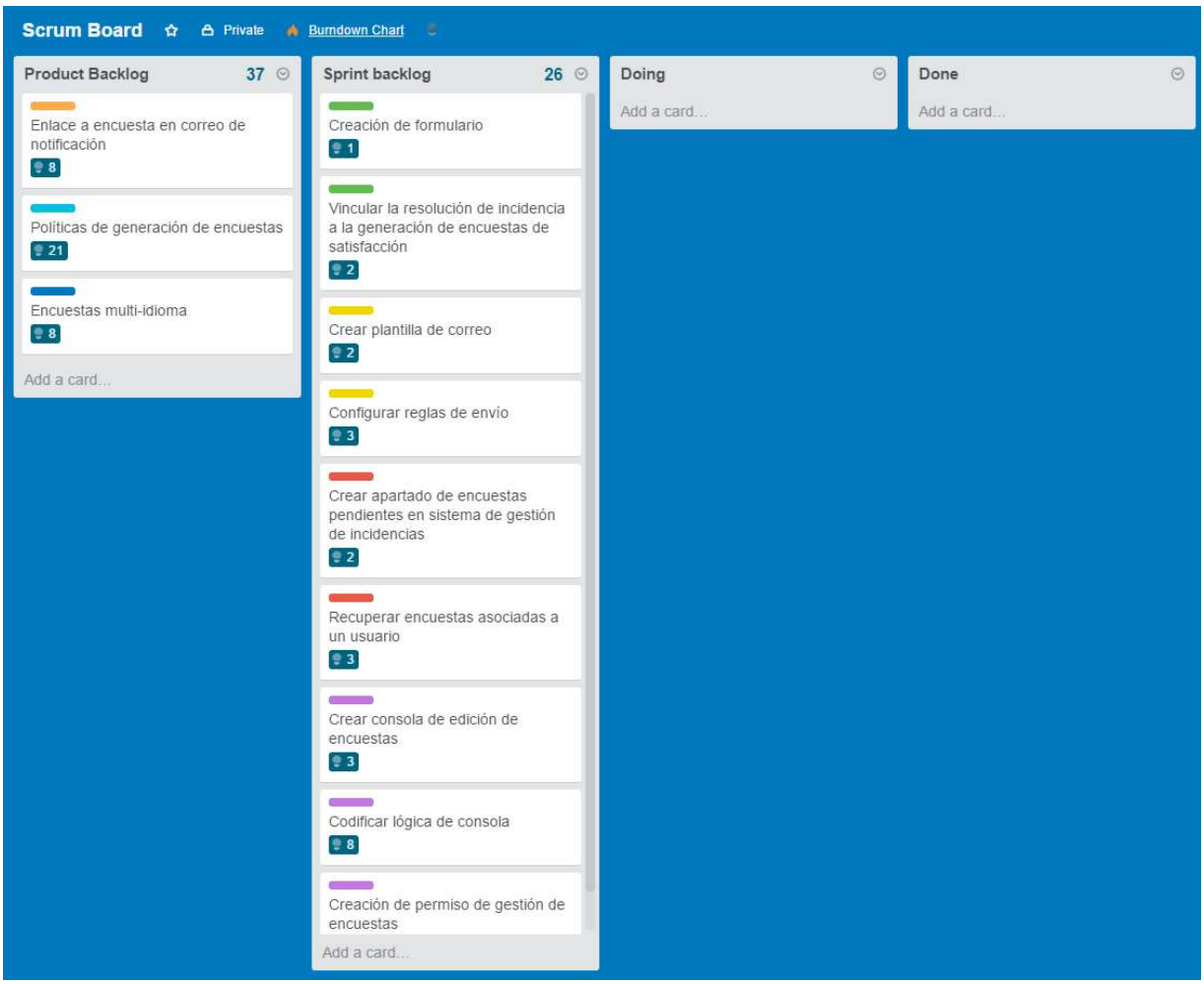


Figura 33. Descomposición de historias de usuario en tareas del sprint 1

Una vez tenemos las tareas a realizar, podemos comentar la primera iteración. La gráfica de la Figura 34 muestra el trabajo restante después de cada iteración, que en este caso sería el total de los puntos de historia identificados en el proyecto.

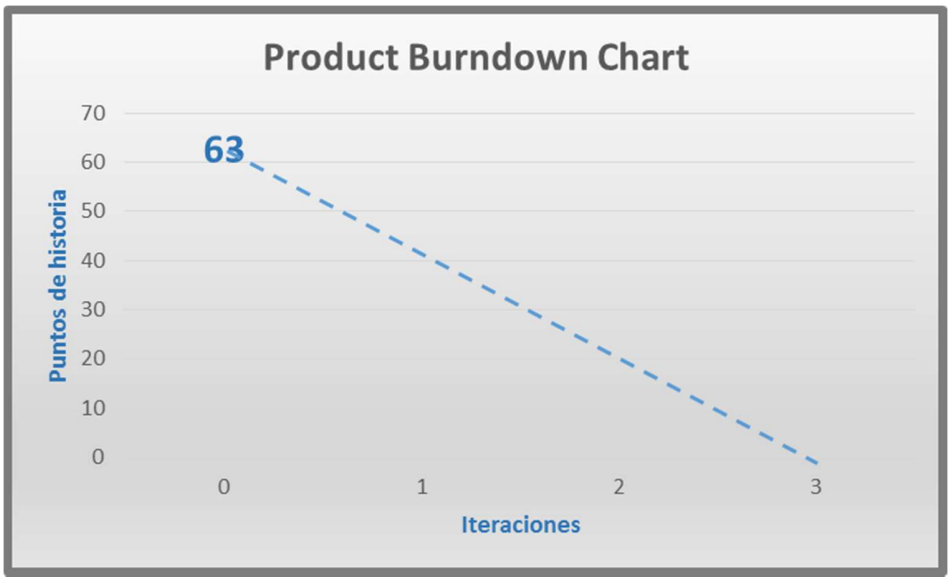


Figura 34. Product Burndown Chart antes de comenzar las iteraciones

4.4. Sprint 1

Reunión diaria

El equipo se reúne delante del panel y responde a tres preguntas: ¿Qué has hecho ayer? ¿Qué voy a hacer hoy? ¿Algún impedimento?

- **Jornada 1**

Como se trata de la primera reunión los integrantes del equipo únicamente deciden que tareas van a realizar en esa jornada.

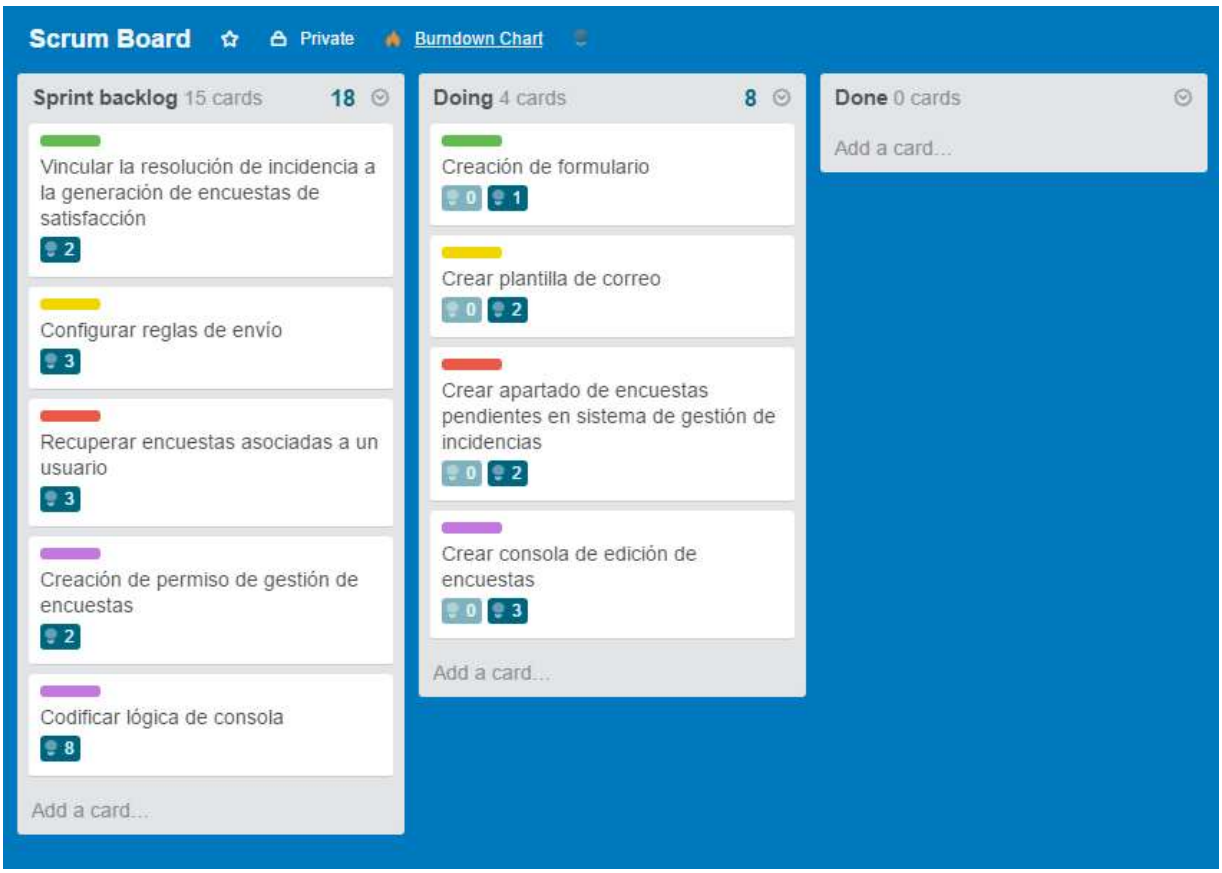


Figura 35. Panel Scrum después de la jornada 1 del sprint 1

El gráfico de la Figura 36 ilustra el trabajo restante después de cada jornada del *sprint*.

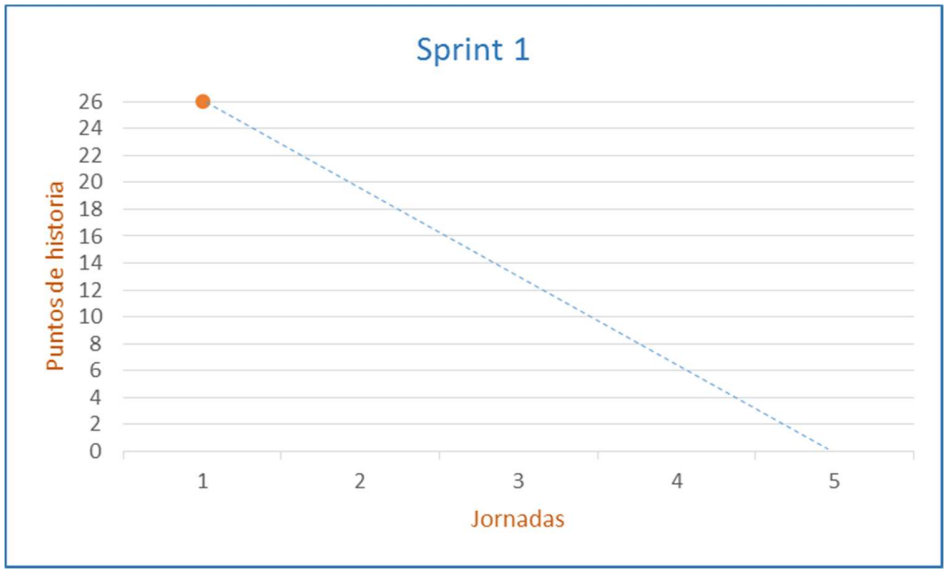


Figura 36. Sprint Burndown Chart en la jornada 1 del sprint 1

➢ **Jornada 2**

El equipo responde las tres preguntas, lo que implica actualizar el trabajo realizado y coger nuevas tareas para hacer durante la jornada. La Figura 37 ilustra el panel en la jornada 2 del primer *sprint*.

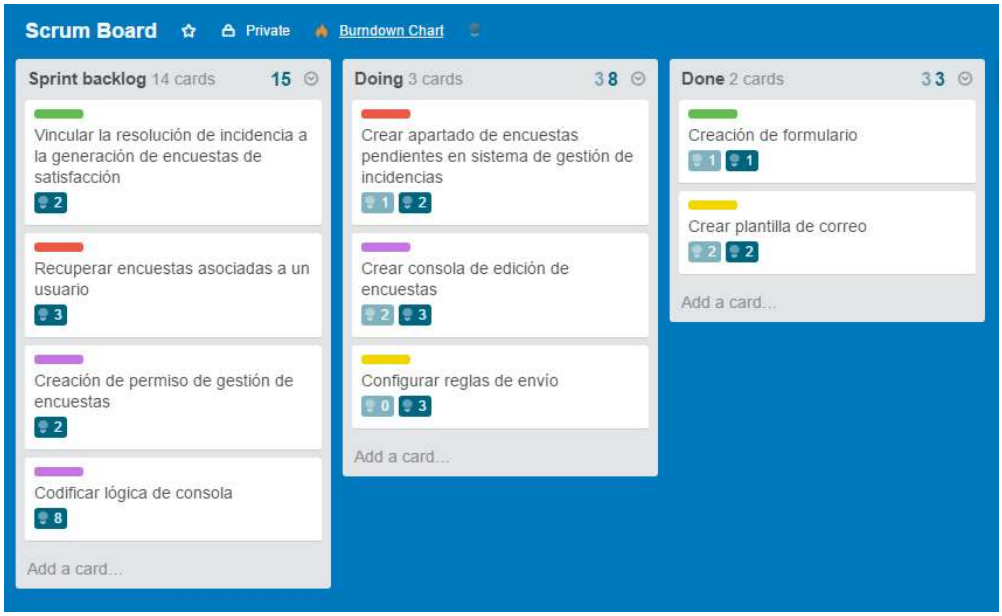


Figura 37. Panel Scrum después de la jornada 2 del sprint 1

La Figura 38 ilustra el trabajo restante después de la jornada 2.

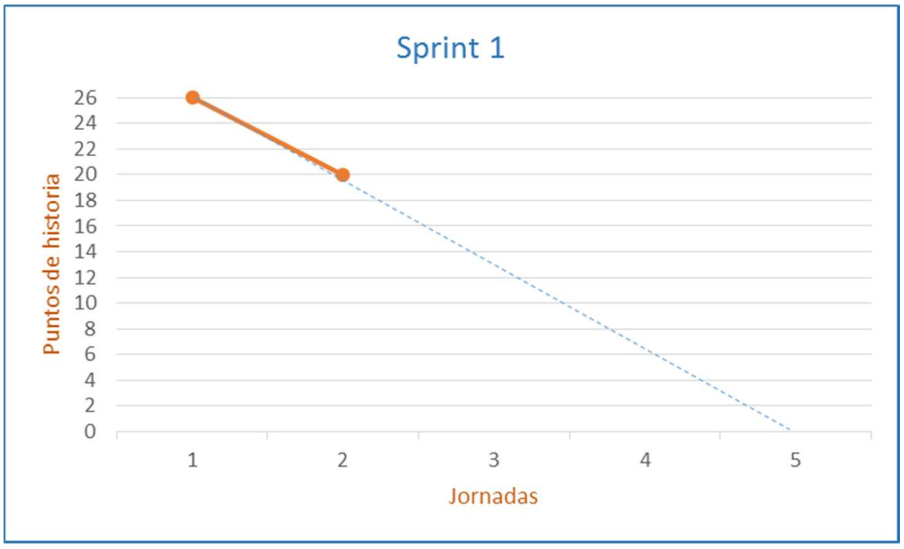


Figura 38. Sprint Burndown Chart en la jornada 2 del sprint 1

- **Jornada 3**

El equipo responde a las tres preguntas y actualiza el panel, dando como resultado la Figura 39.

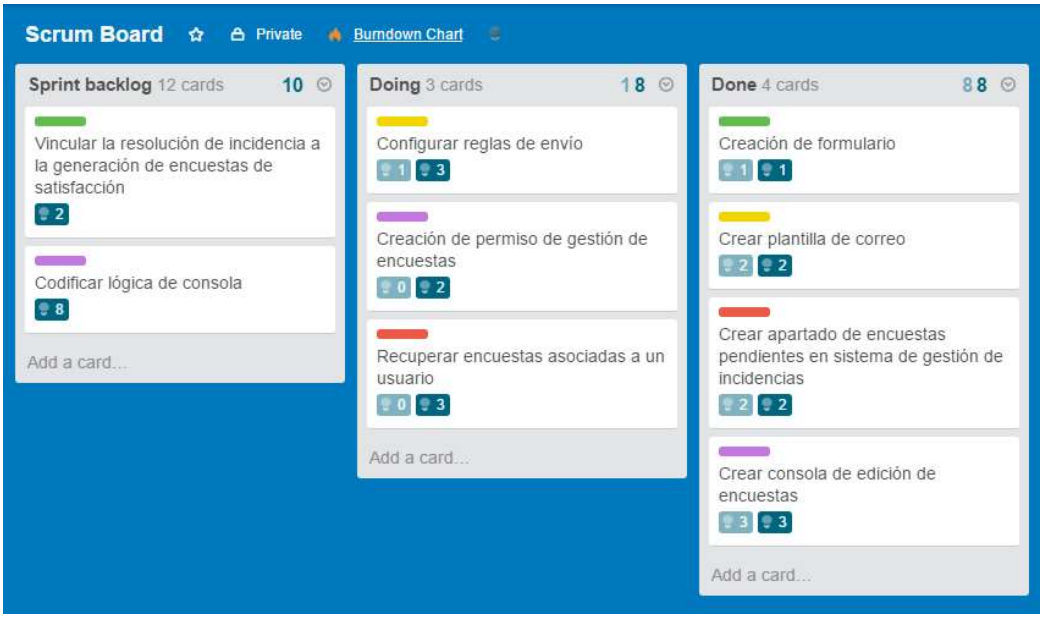


Figura 39. Panel Scrum después de la jornada 3 del sprint 1

En la Figura 40 se observa como el desarrollo avanza más lentamente de lo previsto.

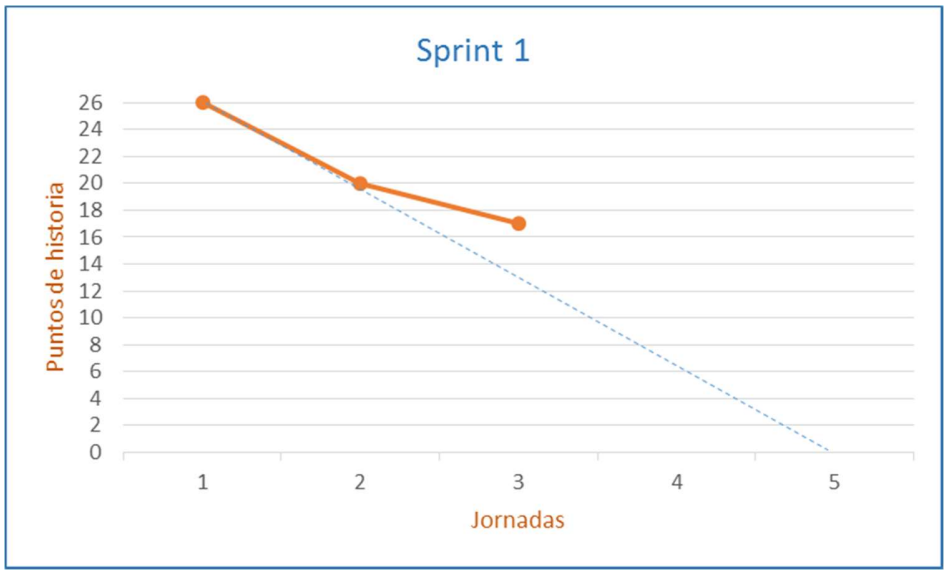


Figura 40. Sprint Burndown Chart en la jornada 3 del sprint 1

➢ **Jornada 4**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 41.

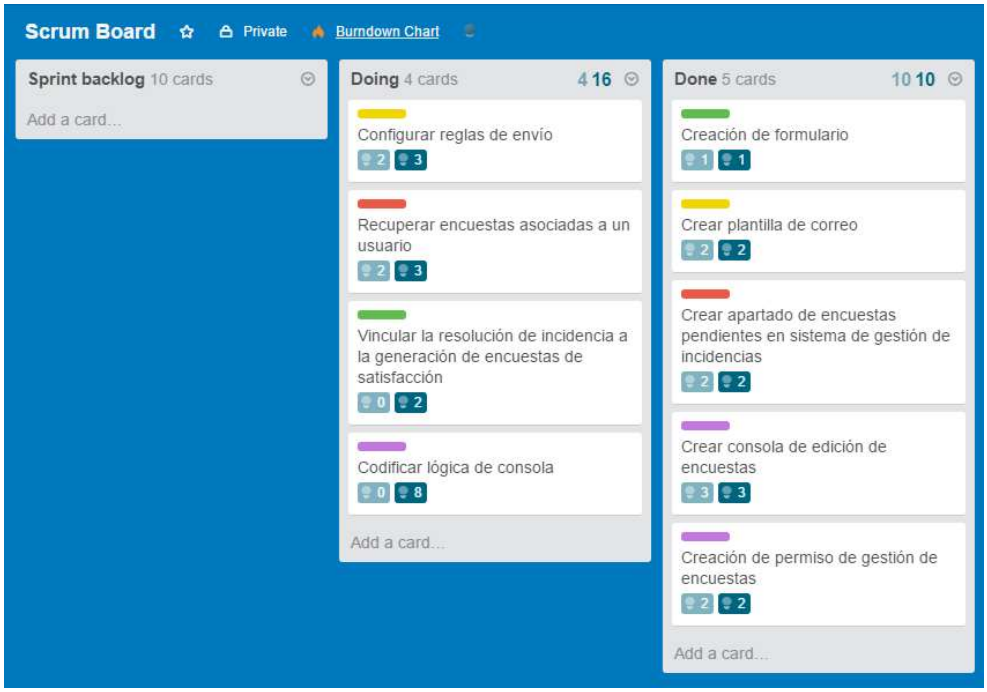


Figura 41. Panel Scrum después de la jornada 4 del sprint 1

Observando la Figura 42 se confirma la tendencia detectada en la jornada anterior.

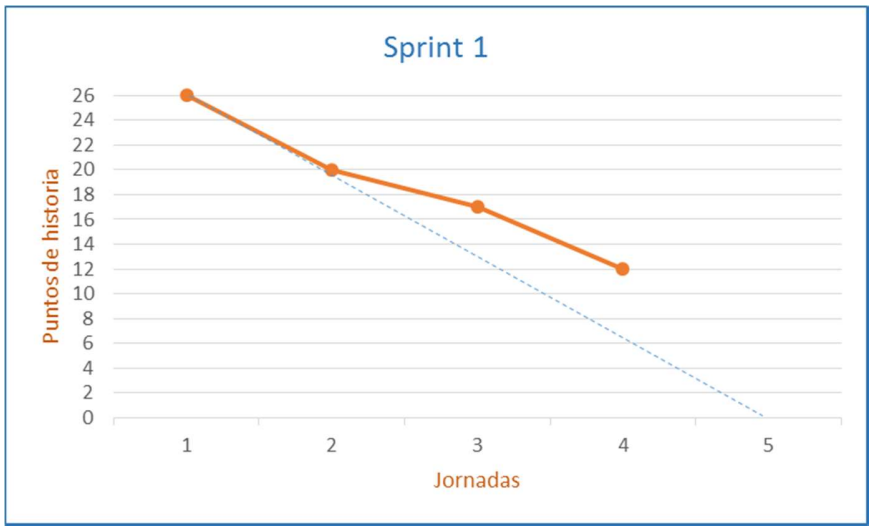


Figura 42. Sprint Burndown Chart en la jornada 4 del sprint 1

➢ **Jornada 5**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 43.

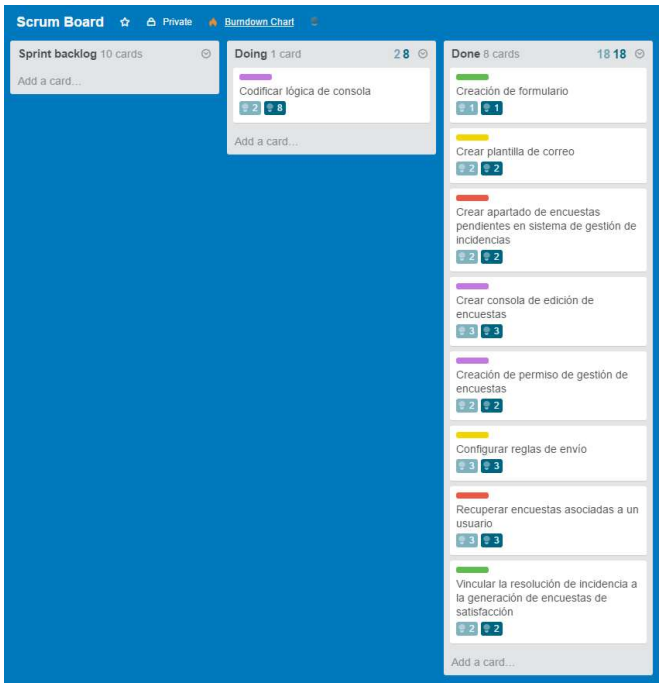


Figura 43. Panel Scrum después de la jornada 5 del sprint 1

Al finalizar la primera iteración el equipo no ha podido finalizar todas las historias de usuario comprometidas tal y como se puede apreciar en la Figura 44. Una de las tareas de la historia de usuario está incompleta y por tanto, la historia de usuario no puede ser considerada como completa. Esto supone que se han completado **13 puntos de historia** de los 26 comprometido.

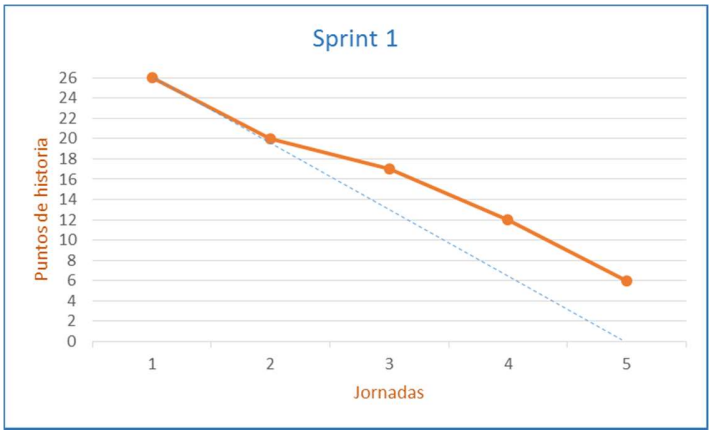


Figura 44. Sprint Burndown Chart en la jornada 5 del sprint 1

Este resultado podría verse como malo, pero al tratarse de la primera iteración de un equipo novato en *Scrum* es normal que el equipo haya sobreestimado su capacidad para completar historias de usuario. Otro posible motivo de este resultado sea una historia de usuario demasiado grande, que podría haberse dividido en historias de usuario menores.

La gráfica mostrada en la Figura 45 muestra los puntos de historia restantes para completar el proyecto completo. En esta gráfica no se consideran los puntos de historia de las tareas de la historia de usuario incompleta, es decir, sólo se consideran los puntos de historia de las historias de usuario completas.

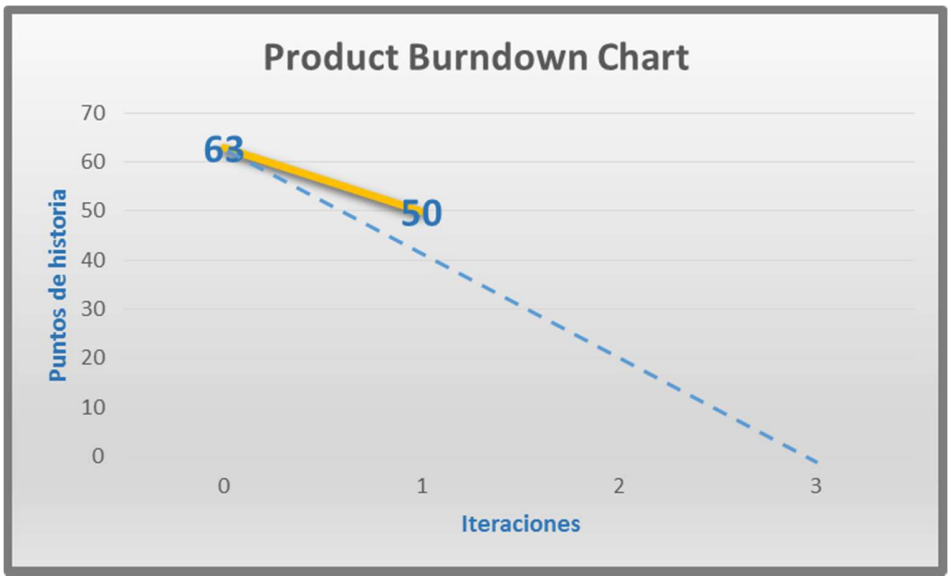


Figura 45. Product Burndown Chart después de la iteración 1

4.5. Sprint 2

Antes de comenzar la segunda iteración el equipo selecciona las historias de usuario que se compromete a entregar. En este caso también se incluye la historia de usuario que no dio tiempo a terminar en la primera iteración (Figura 46).

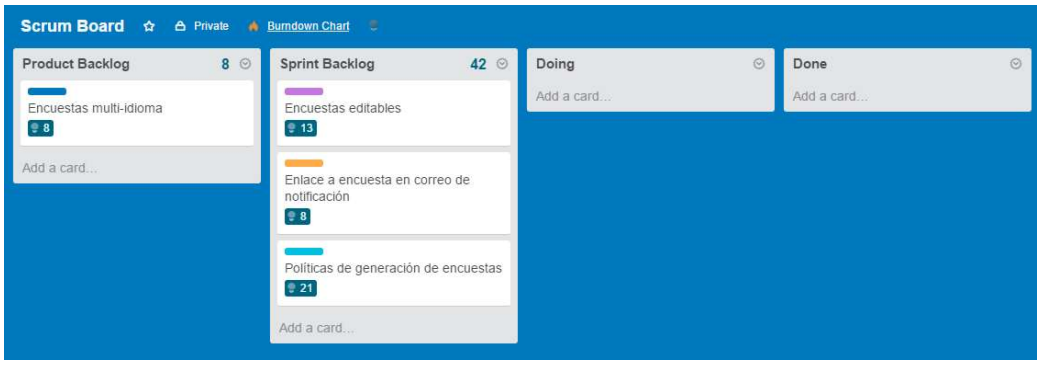


Figura 46. Selección de historias de usuario del sprint 2

El equipo descompone en tareas las historias de usuario seleccionadas, dando como resultado una Pila de la Iteración (Sprint Backlog) como el de la Figura 47.

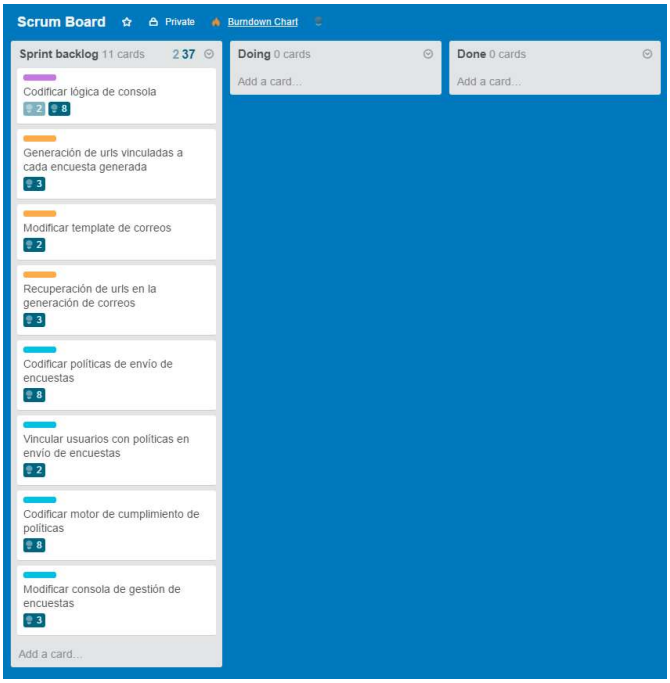


Figura 47. Sprint Backlog del sprint 2

➢ **Jornada 1**

El equipo se reúne delante del panel y responde a las tres preguntas: ¿Qué has hecho ayer? ¿Qué voy a hacer hoy? ¿Algún impedimento?

Como se trata de la primera reunión del *sprint 2* los integrantes del equipo únicamente deciden que tareas van a realizar en esa jornada y actualizan el panel, resultando un panel como el de la Figura 48.

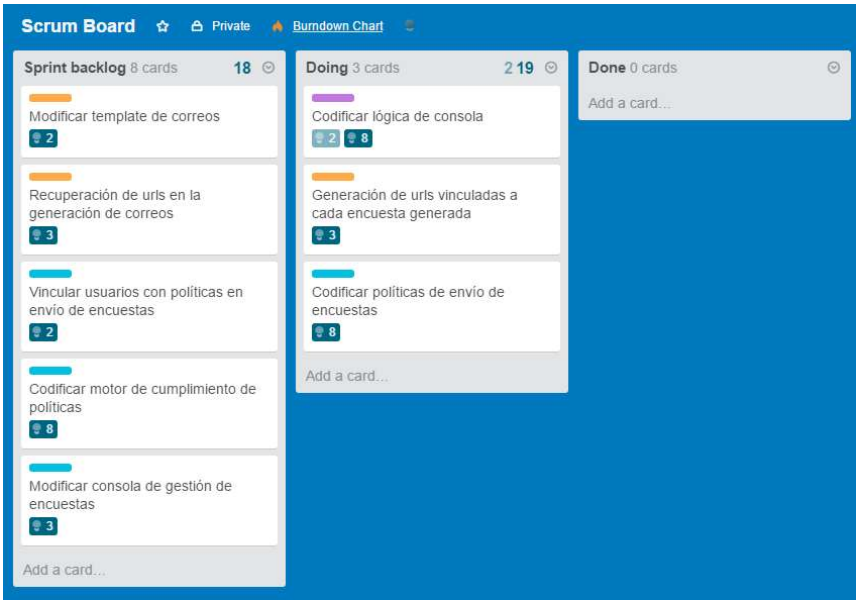


Figura 48. Panel Scrum después de la jornada 1 del sprint 2

Al inicio del sprint 2 faltan por completarse todos los puntos de historia seleccionados por el equipo, tal y como puede observarse en la Figura 49.

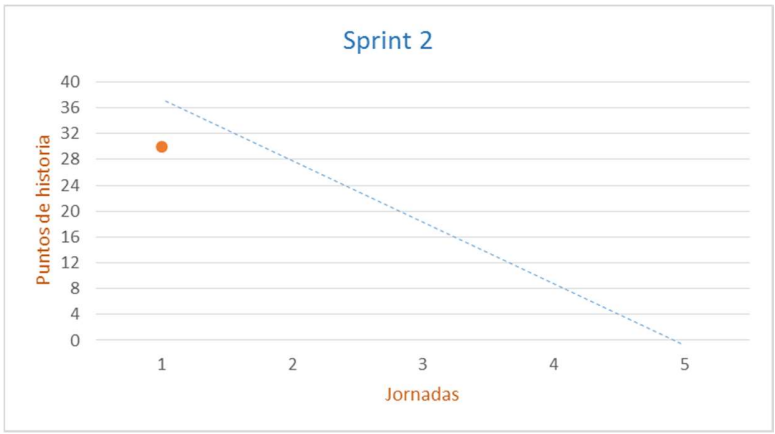


Figura 49. Sprint Burndown Chart en la jornada 1 del sprint 2

- **Jornada 2**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 50.

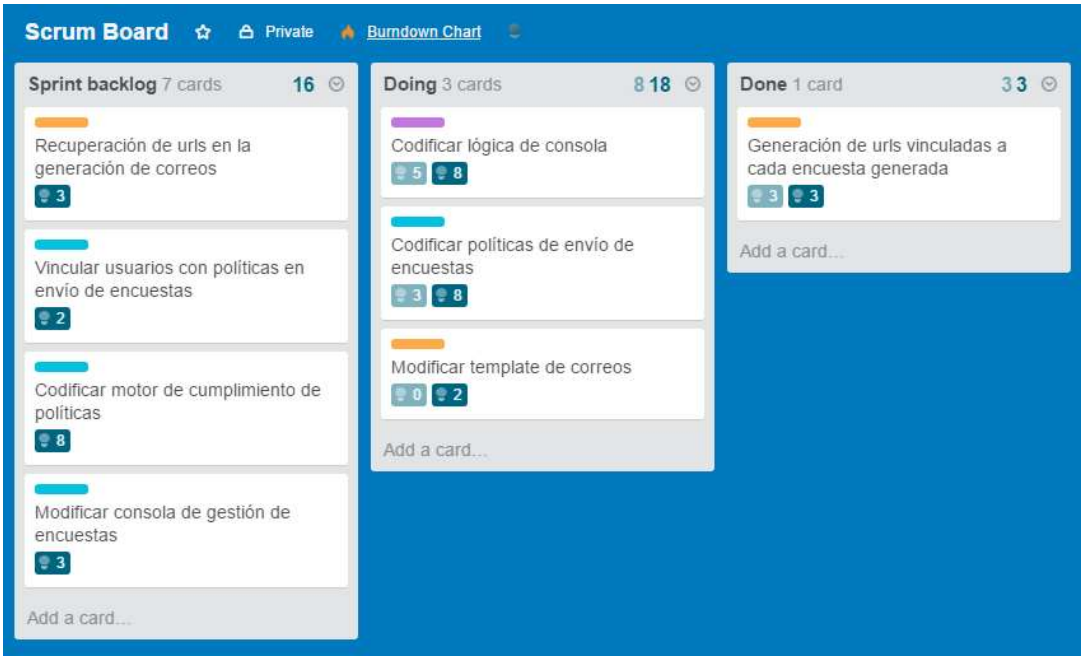


Figura 50. Panel Scrum después de la jornada 2 del sprint 2

La gráfica de la Figura 51 muestra un ligero adelanto de los desarrollos con respecto al avance teórico.

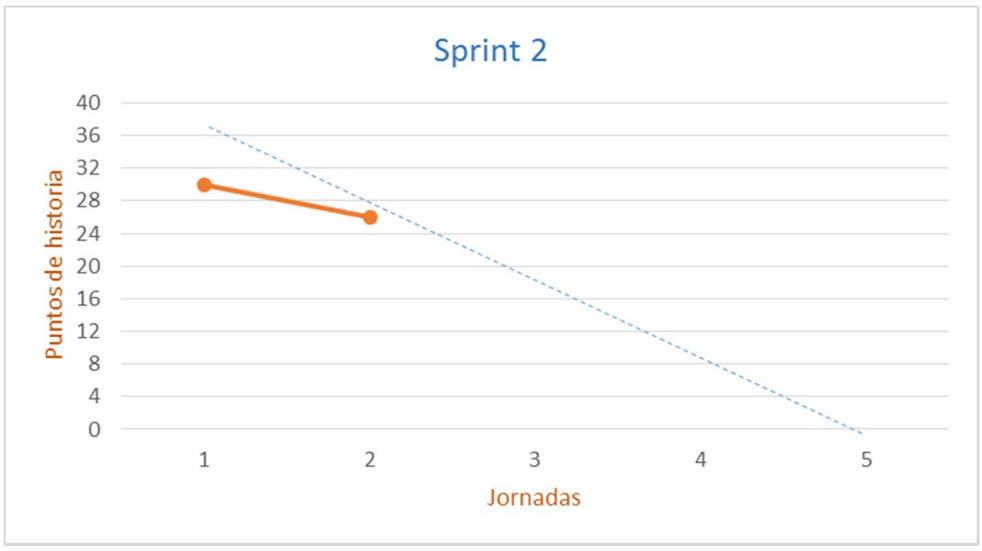


Figura 51. Sprint Burndown Chart en la jornada 2 del sprint 2

➢ **Jornada 3**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 52.

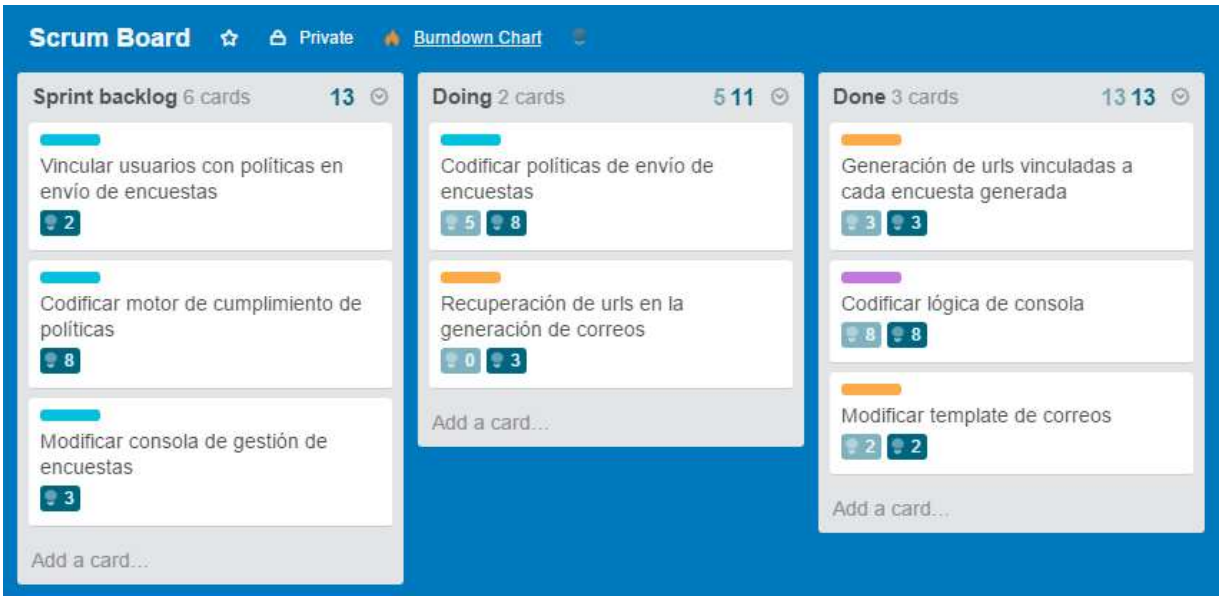


Figura 52. Panel Scrum después de la jornada 3 del sprint 2

El avance de los desarrollos sigue según lo esperado, tal y como se observa en la Figura 53.

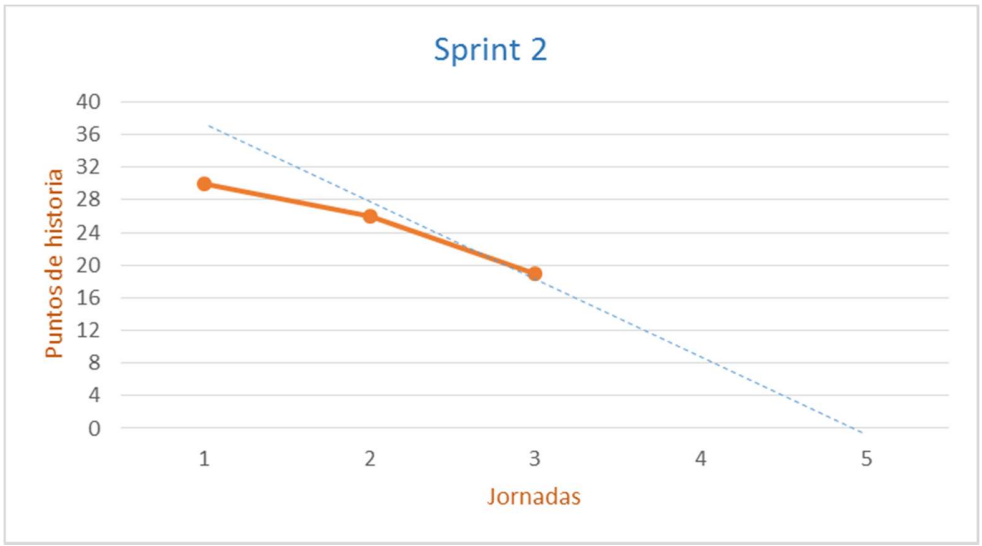


Figura 53. Sprint Burndown Chart en la jornada 3 del sprint 2

➢ **Jornada 4**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 54.

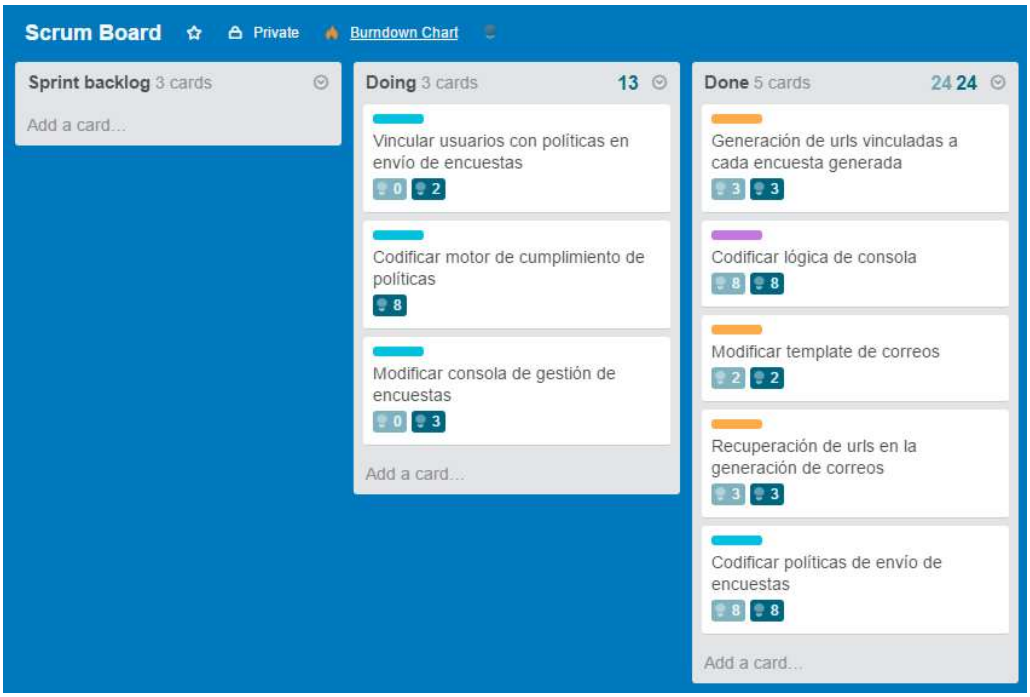


Figura 54. Panel Scrum después de la jornada 4 del sprint 2

En la Figura 55 se observa una desviación con respecto a lo planificado. En la próxima jornada se confirmará o no si va a suponer no entregar las todas las historias de usuario comprometidas.

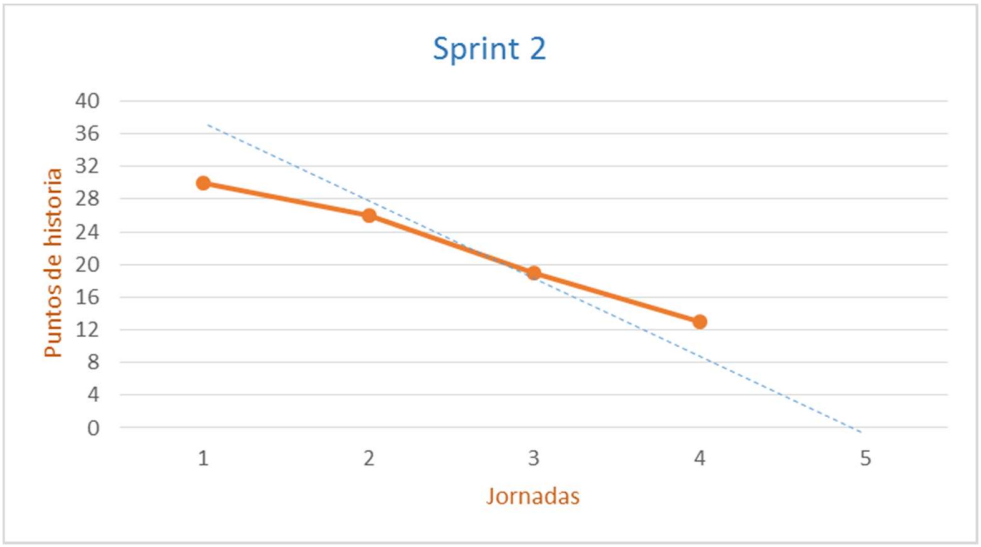


Figura 55. Sprint Burndown Chart en la jornada 4 del sprint 2

- **Jornada 5**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 56.

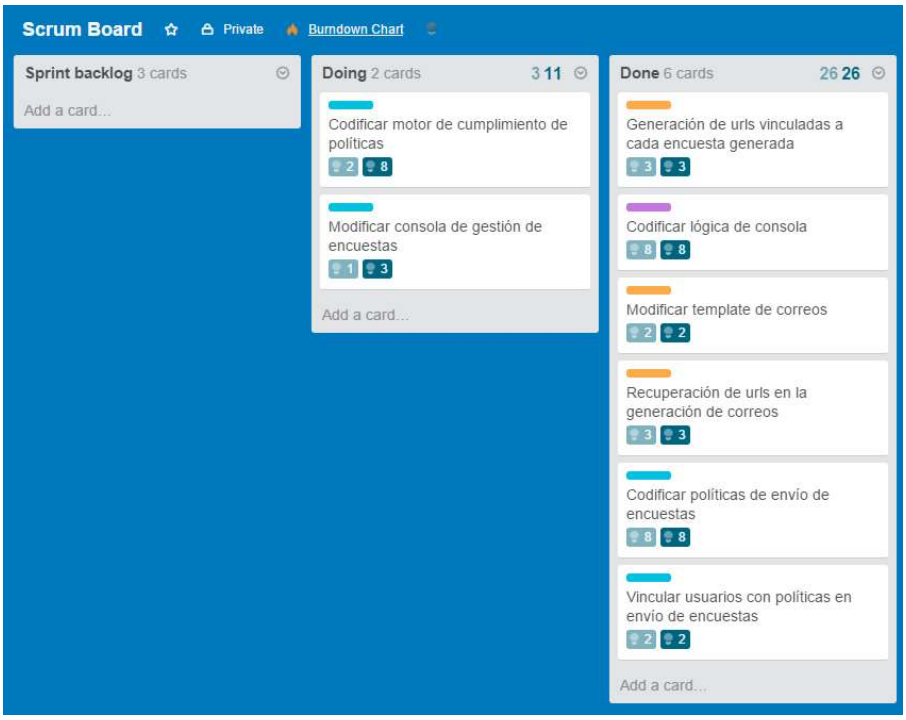


Figura 56. Panel Scrum después de la jornada 5 del sprint 2

Se confirma la tendencia y faltan por completarse 8 puntos de historia comprometidos, tal y como se observa en la Figura 57.

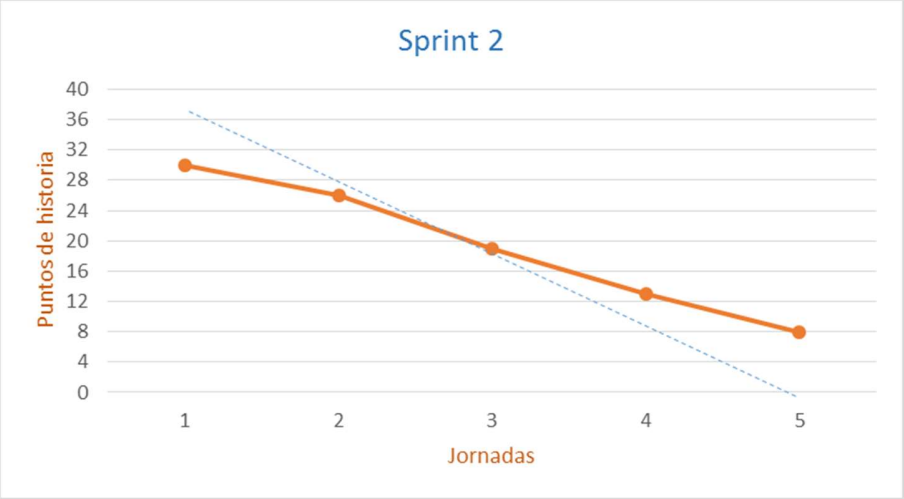


Figura 57. Sprint Burndown Chart en la jornada 5 del sprint 2

Los 8 puntos de historia sin completar se traducen en que la historia de usuario al completo no puede entregarse. La tendencia parece indicar que los desarrollos no finalizarán en la siguiente iteración (Figura 58).

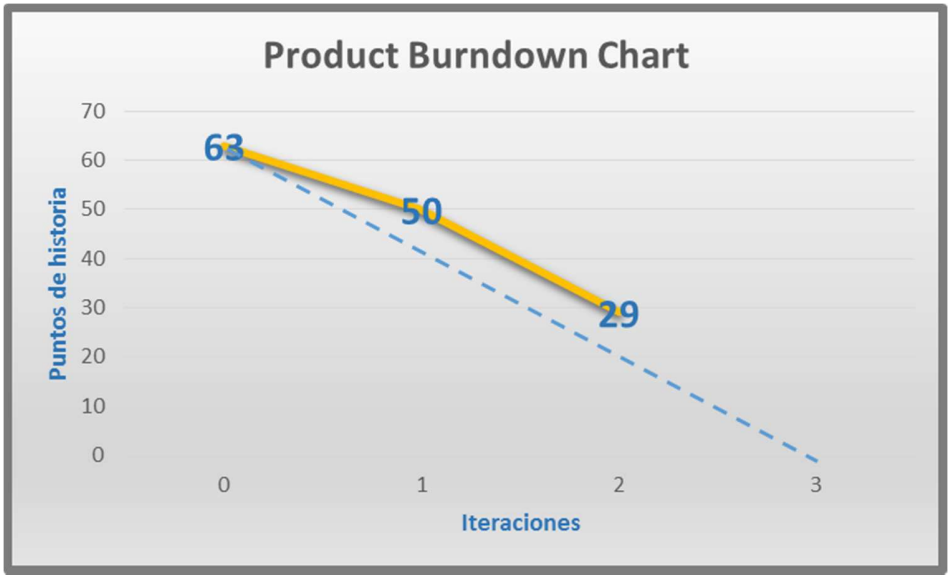


Figura 58. Product Burndown Chart después de la iteración 2

4.6. Sprint 3

Antes de comenzar la tercera iteración el equipo selecciona las historias de usuario que se compromete a entregar. En este caso sólo queda una historia de usuario más la historia de usuario que no se completó en la segunda iteración (Figura 59).

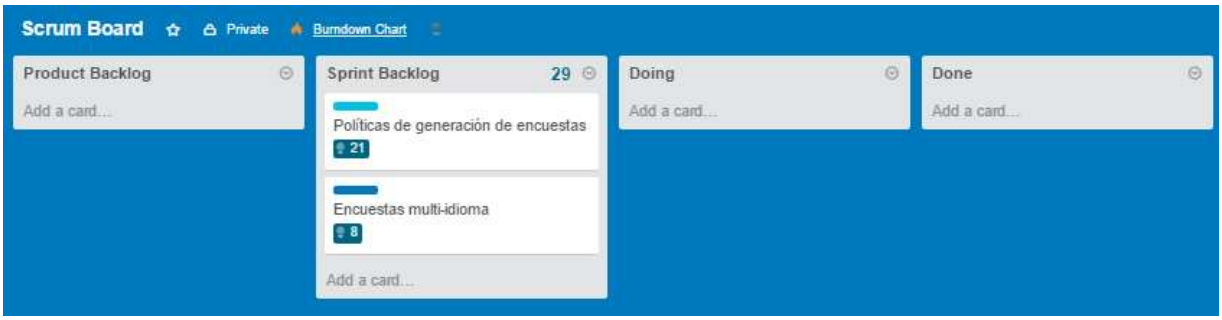


Figura 59. Selección de historias de usuario del sprint 3

El equipo descompone en tareas las historias de usuario seleccionadas, dando como resultado una Pila de la Iteración (*Sprint Backlog*) como el de la Figura 60.

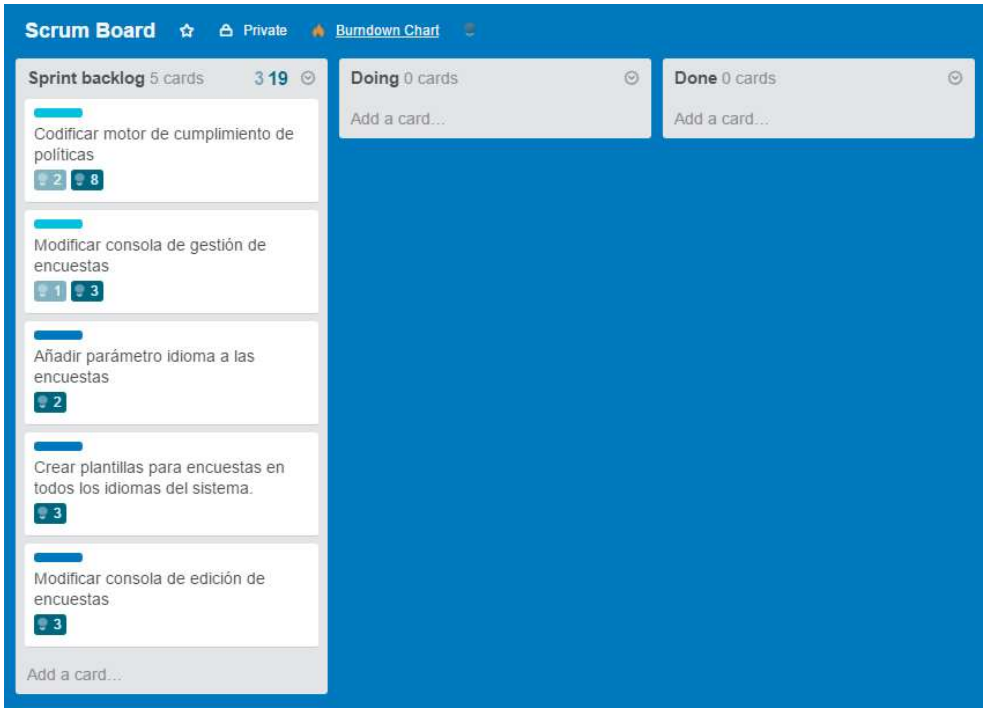


Figura 60. Sprint Backlog del sprint 3

➢ **Jornada 1**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 61.

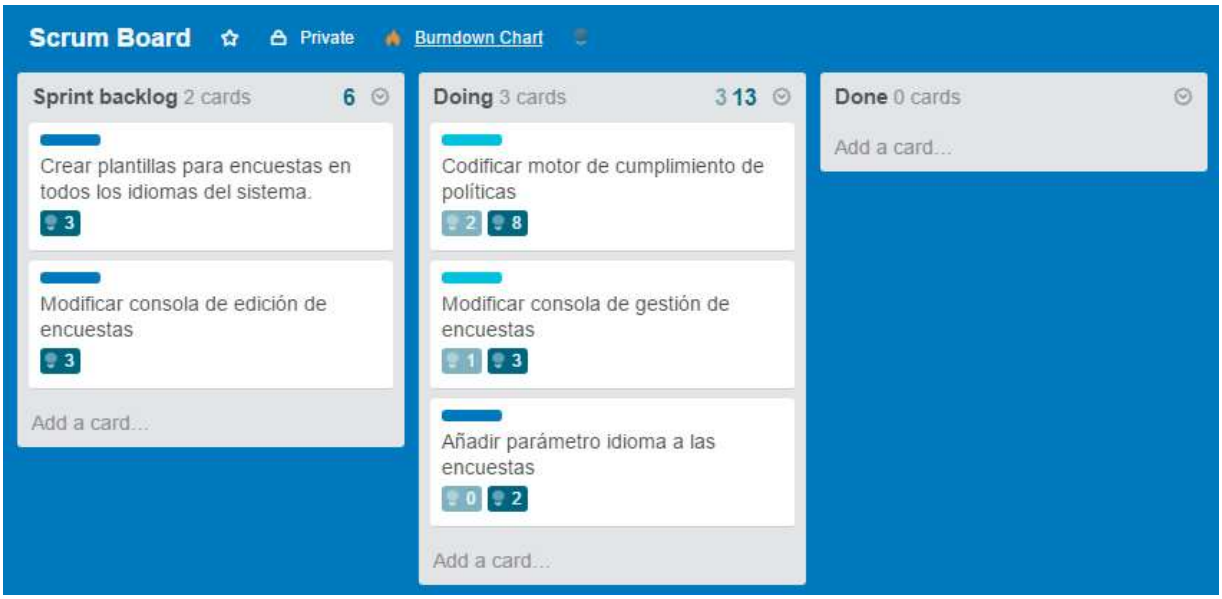


Figura 61. Panel Scrum después de la jornada 1 del sprint 3

En este caso al inicio de la iteración ya hay completados 3 puntos de historia que se completaron en la iteración anterior (Figura 62).

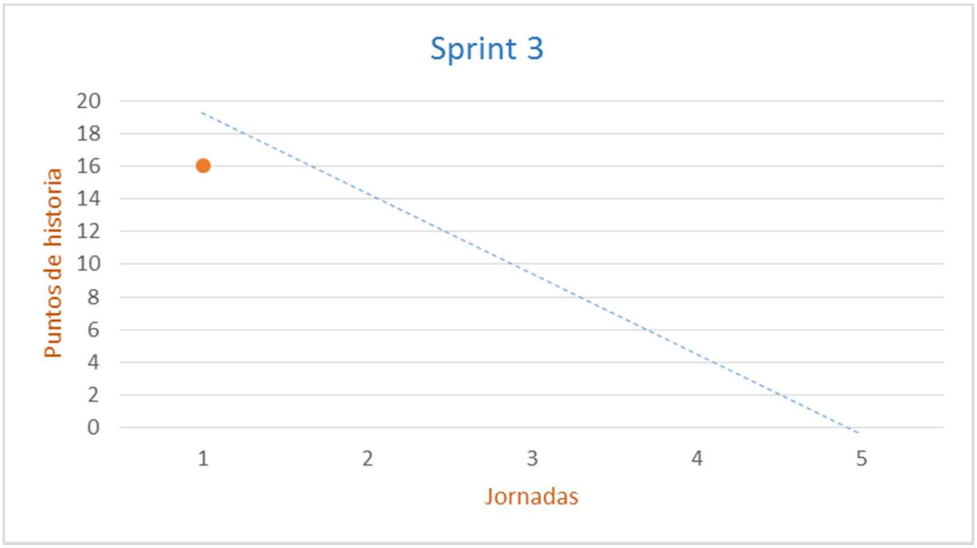


Figura 62. Sprint Burndown Chart en la jornada 1 del sprint 3

- **Jornada 2**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 63.

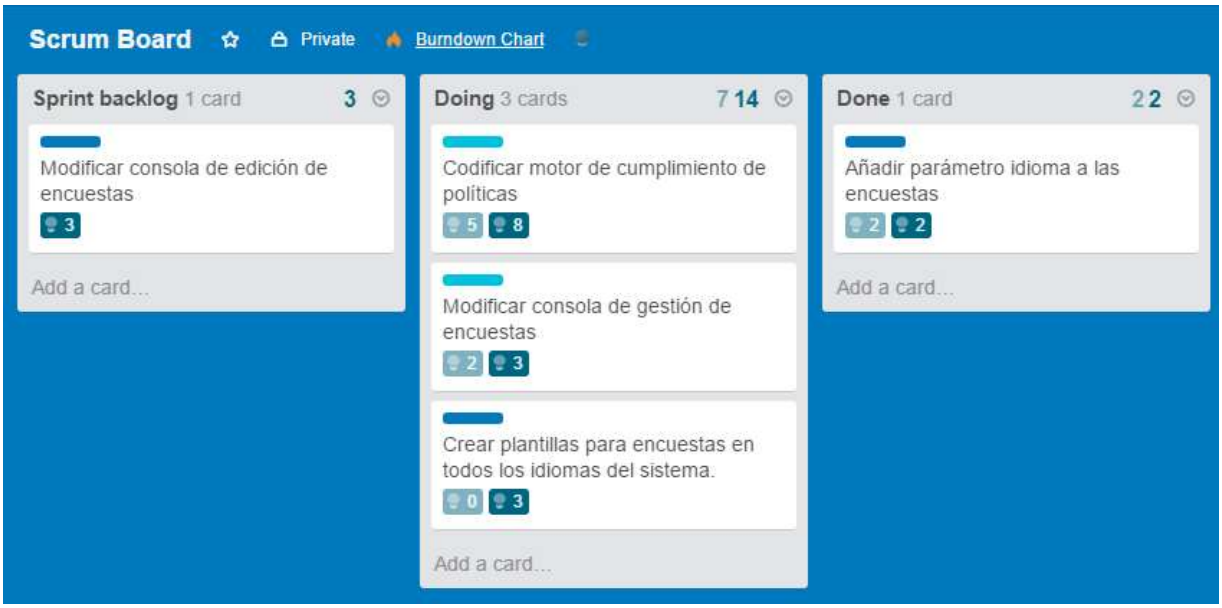


Figura 63. Panel Scrum después de la jornada 2 del sprint 3

La gráfica de la Figura 64 muestra una tendencia muy positiva de cara a finalizar todos los puntos de historia comprometidos.

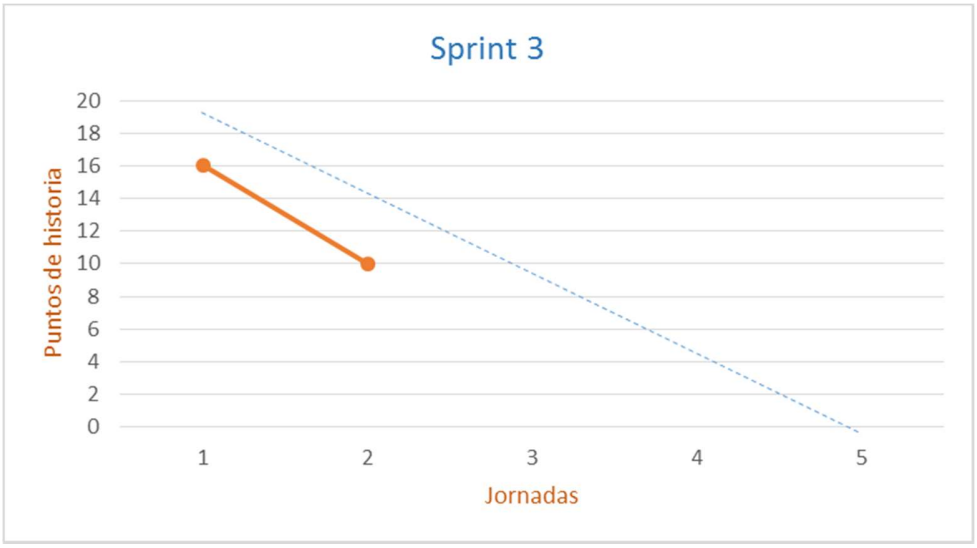


Figura 64. Sprint Burndown Chart en la jornada 2 del sprint 3

➢ **Jornada 3**

El equipo responde las tres preguntas y actualiza el panel, dando como resultado la Figura 65.

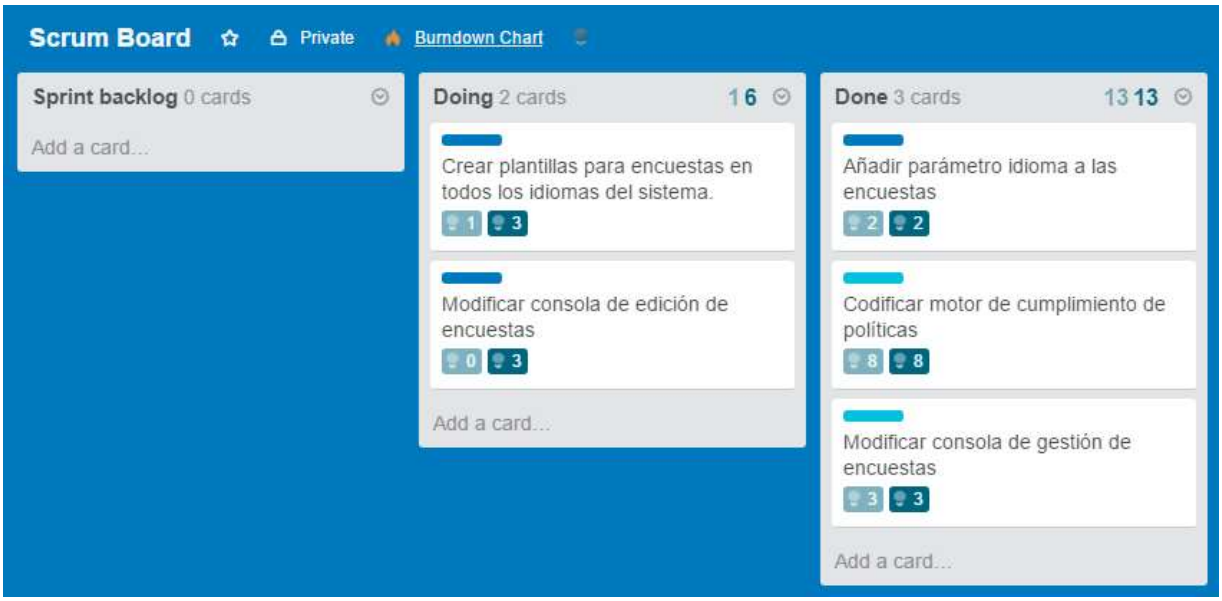


Figura 65. Panel Scrum después de la jornada 3 del sprint 3

Si no surgen imprevistos, podrían finalizarse los desarrollos antes de lo previsto (Figura 66).

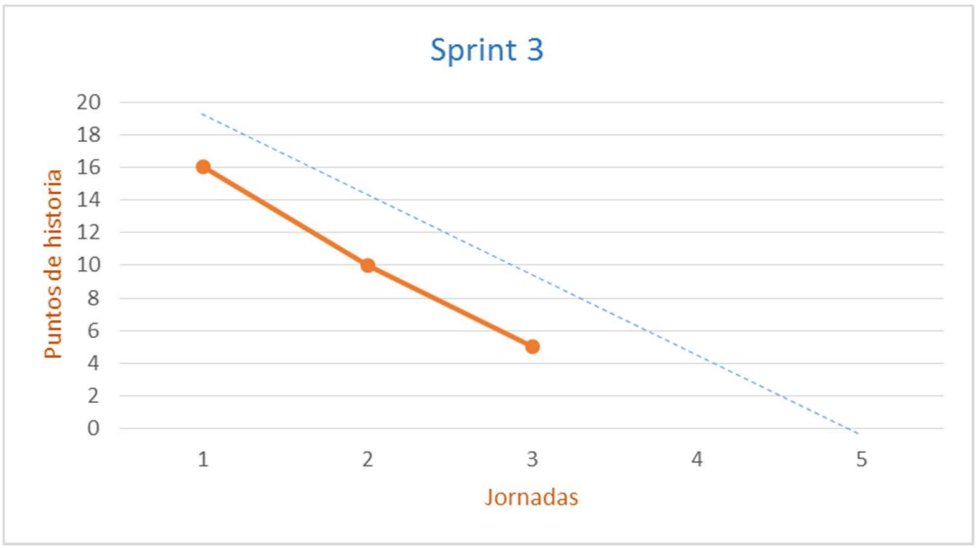


Figura 66. Sprint Burndown Chart en la jornada 3 del sprint 3

- **Jornada 4**

El equipo actualiza el panel y todas las tareas están finalizadas.

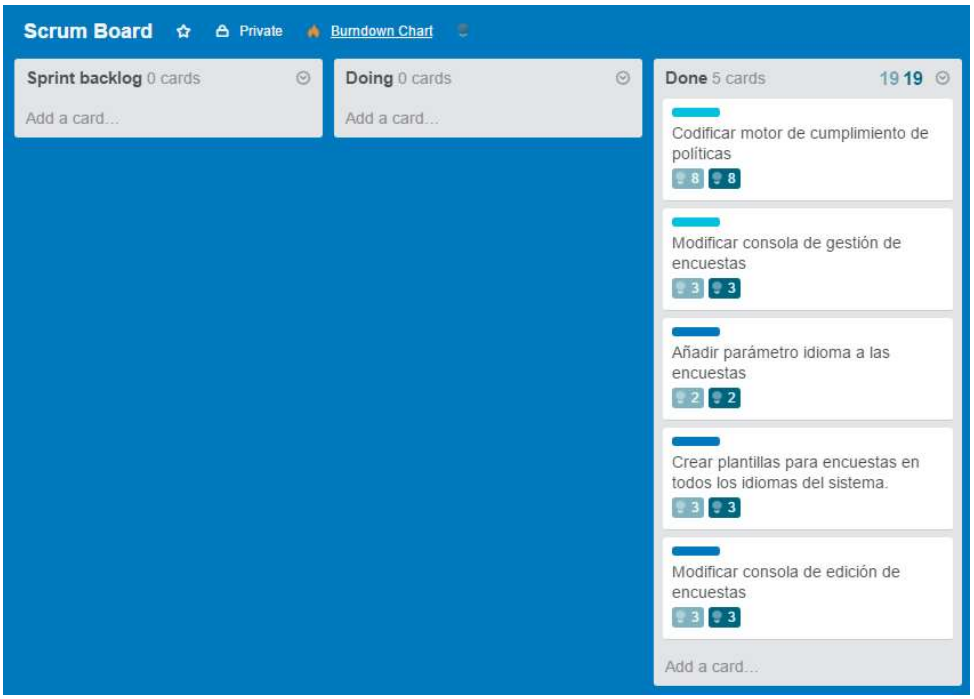


Figura 67. Panel Scrum después de la jornada 4 del sprint 3

La Figura 68 muestra como el ritmo de desarrollo ha sido constante durante esta iteración. Los puntos de historia que quedaban para esta iteración eran inferiores a la capacidad del equipo.

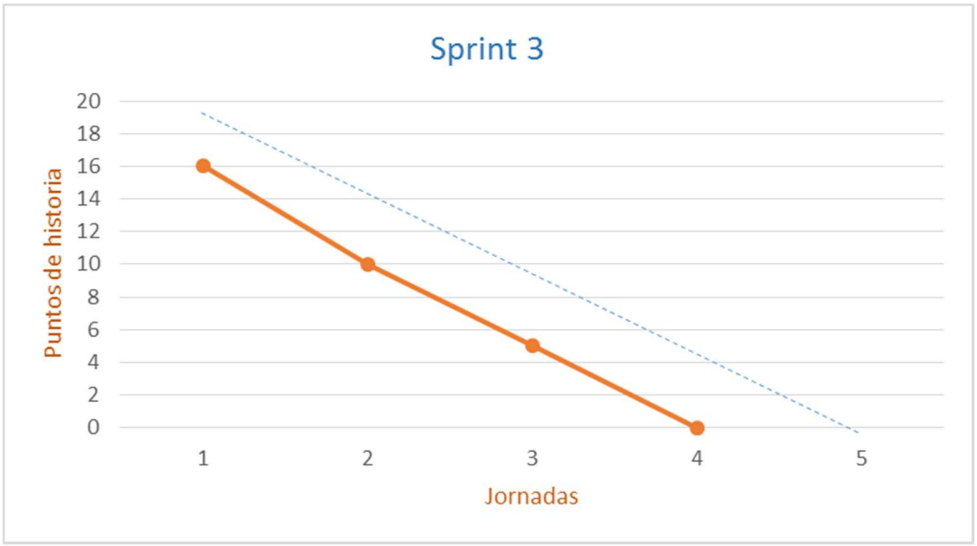


Figura 68. Sprint Burndown Chart en la jornada 4 del sprint 3

Después de esta iteración se han entregado 29 puntos de historia correspondientes a las 2 últimas historias de usuario que faltaban por completar, tal y como se observa en la Figura 69. Sin embargo, en realidad solo han desarrollado 16 puntos de historia dado que los 13 restantes se desarrollaron en la segunda iteración. Al no completarse la historia de usuario sus puntos de historia no se contabilizaron en esa iteración y se contabilizan en esta última.

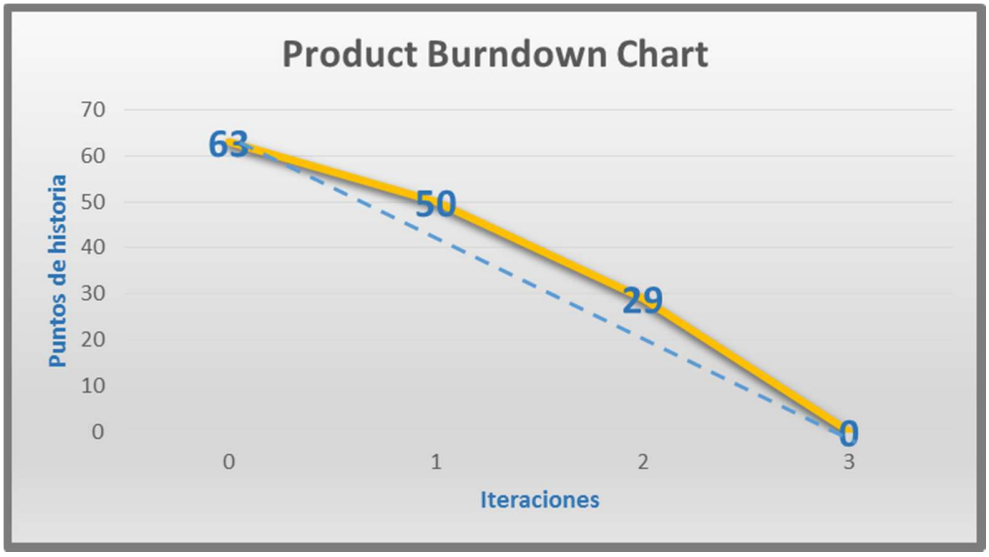


Figura 69. Product Burndown Chart después de la iteración 3

Capítulo 6

Valoración económica

En este apartado se detallan los costes teóricos de la realización del proyecto. Para realizar dicho cálculo se analizan dos aspectos: recursos humanos y recursos materiales.

6.1. Recursos humanos

En este proyecto se han empleado únicamente dos recursos, un coordinador de las actividades y un investigador, durante un período aproximado de 7 meses a tiempo parcial. Para calcular el coste que supone la actividad de dichos recursos se han desglosado las tareas y se les han asignado una duración en horas invertidas por parte del investigador. Para calcular las horas dedicadas por parte del coordinador se ha calculado el 10% de las horas invertidas por el investigador. Con dichas cifras calculadas en horas se ha obtenido la dedicación en meses, suponiendo que cada mes está compuesto de 131.25 horas. Finalmente se ha multiplicado el esfuerzo calculado por el coste mensual de cada uno de los recursos.

Tarea	Horas
Capítulo 1	**12**
Capítulo 2	**30**
Capítulo 3	**98**
Selección de metodologías	10
Scrum	30
eXtreme Programming	17
Crystal	17
Kanban	12
Scrumban	12
Capítulo 4	**35**
Capítulo 5	**25**
Capítulo 6	**15**
Capítulo 7	**16**
Maquetación	**12**
Correcciones	**45**
Total	**288**
Meses dedicación	**2,2**

Tabla 19. Desglose de tareas y horas del proyecto

Apellidos y nombre	N.I.F. (no rellenar - solo a título informativo)	Categoría	Dedicación (hombres mes)	Coste hombre mes	Coste (Euro)
Raquel Pérez Leal		Ingeniero Senior	0,22	4.289,54	943,70
Jorge González del Río		Ingeniero	2,2	2.694,39	5.927,66
		Hombres mes 2,42		**Total**	6.871,36

Tabla 20. Costes de recursos humanos

6.2. Recursos materiales

En este apartado se calcula el coste aproximado de los recursos materiales empleados para la realización del proyecto. En este apartado se tienen en cuenta los costes de amortización de los equipos utilizados durante la realización del proyecto. Dado que este proyecto es teórico y el

caso práctico expuesto no se incluye dentro del coste del mismo, el coste de los recursos materiales empleados es el siguiente:

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Ordenador	900,00	50	6	60	45,00
Impresora	150,00	50	6	60	7,50
				Total	52,50

Tabla 21. Costes de equipos materiales

Otros gastos directos son:

Descripción	Empresa	Costes imputable
Consumibles		40,00
	Total	40,00

Tabla 22. Otros gastos materiales directos

6.3. Resumen de costes

Con los datos de los apartados anteriores, y aplicando unos costes indirectos del 20% sobre dichas cantidades, se estima que el coste de la realización de este proyecto es de **8.528 €**.

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	6.871
Amortización	53
Subcontratación de tareas	0
Costes de funcionamiento	40
Costes Indirectos	1.393
Total	8.357

Tabla 23. Presupuesto costes totales

Anexos

A1. Historias de usuario

➢ ¿Qué son las historias de usuario?

Las historias de usuario son un instrumento para el levantamiento de requerimientos para el desarrollo de un software, que ha emergido con la aparición de los nuevos marcos de trabajo de desarrollo ágil, como por ejemplo Scrum o las diferentes técnicas que comprenden el Extreme Programming (XP). Son descripciones cortas de una necesidad de un cliente del software que estemos desarrollando.

➢ ¿Cómo se redactan las historias de usuario?

Al redactar una historia de usuario deben tenerse en cuenta describir el Rol, la funcionalidad y el resultado esperado de la aplicación en una frase corta. Adicionalmente, debe venir acompañada de los criterios de aceptación, no más de 4 por historia, redactado también en una frase que incluya el contexto, el evento y el comportamiento esperado ante ese evento.

➢ ¿Qué características deben tener las historias de usuario?

Algunas características deseables de las historias de usuario son:

- Que sean escritas por el usuario o por un analista de negocio que le represente.
- Frase corta que encaje en una tarjeta de 3 por 5 pulgadas.
- Debe describir el rol desempeñado por el usuario en el sistema, descrito de forma explícita.
- Debe describir el beneficio para el área de negocio que representa esta funcionalidad.

➢ ¿En qué se diferencian de otros instrumentos de levantamiento de requerimientos?

No debemos confundirlas con Casos de Uso o Escenarios de Uso, la gran diferencia es que son más cortas y no deben describir la interfaz con el usuario, los pasos de navegación o flujo de procesos de la aplicación.

➢ ¿Para qué se utilizan?

El propósito principal de esta herramienta es la estimación del esfuerzo necesario para implementar una nueva funcionalidad (Feature), en un software, siguiendo la definición de "Hecho" (DONE) que defina el equipo.

Adicionalmente, se utilizan como iniciadoras de conversaciones entre desarrolladores de software y usuarios del área de negocio, las cuales servirán para identificar los requerimientos del negocio, requerimientos técnicos y para encontrar los supuestos (premisas) no visibles en una primera aproximación. Su propósito, no es describir en detalle la funcionalidad.

Para estas estimaciones, se le asignan unidades de medida a las historias que representen magnitud y no días reales de duración, tales como: puntos de historia, días ideales u otra unidad de medida.

- **¿Cómo se confirma que una historia ha sido implementada adecuadamente y por lo tanto esta "Hecha"?**

Se confirma por medio de la Prueba de Aceptación, la cual a su vez deriva de los criterios de aceptación (también denominados condiciones de satisfacción) asociados a la historia.

- **¿Qué problemas puede ocasionar el acometer errores al escribir las historias de usuario?**

El levantamiento de requerimientos y las pruebas en el desarrollo ágil de software dependen en gran medida de las historias de usuario, por lo que si cometemos errores al definir las mismas podemos tener problemas a lo largo del desarrollo.

Podría pensarse, como cometer errores al expresar requerimientos de una forma tan simple, sin embargo, esta simplicidad puede ocasionar problemas. Por supuesto que el escribir buenas historias de usuario será más difíciles para equipos novatos en metodologías ágiles.

Los errores cometidos al escribir las historias pueden ocasionar un mal entendimiento de los requerimientos, mala formulación de casos de pruebas, y lo que es peor, una mala implementación, ocasionando el rechazo de los entregables de una iteración.

[4]

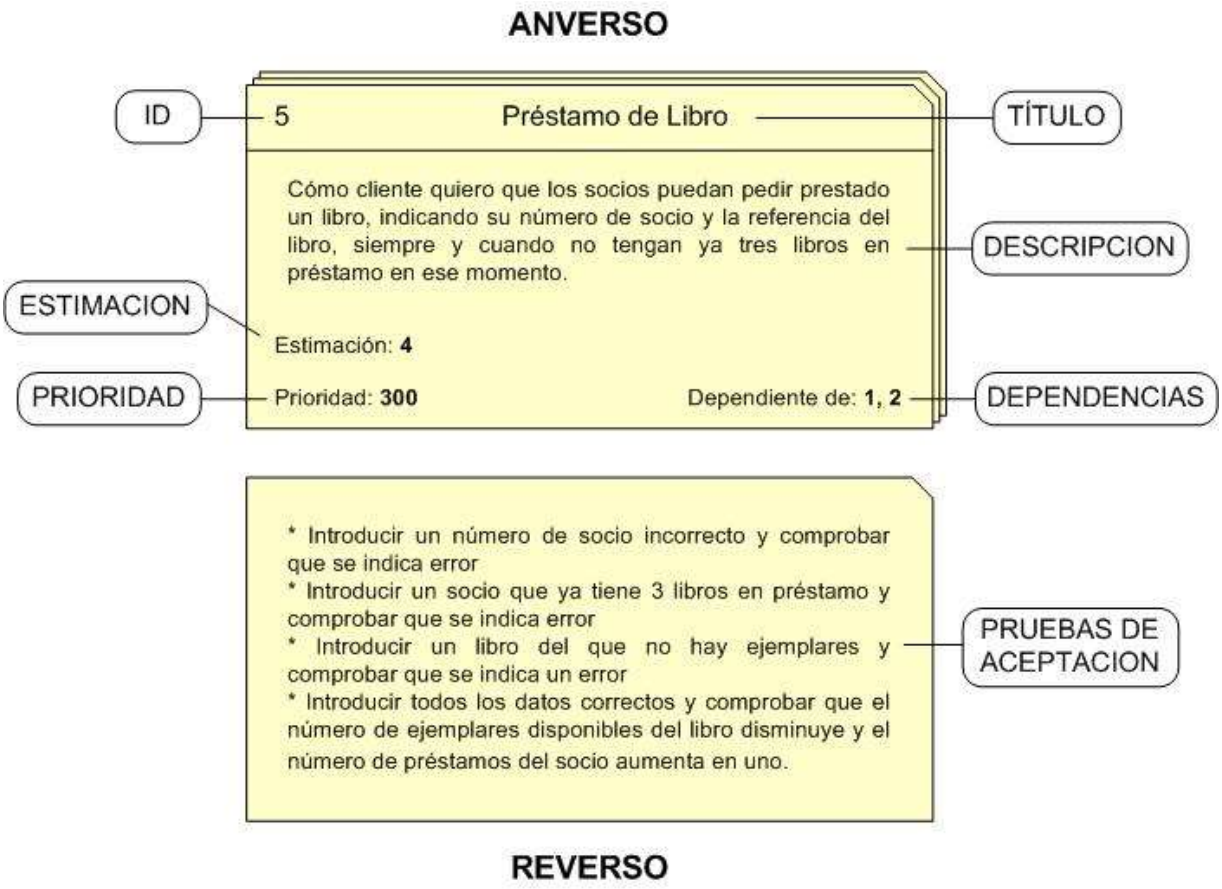


Figura 70. Ejemplo de historia de usuario

A2. Estimación ágil de proyectos software

La estimación es una medida que nos ayuda a calcular el esfuerzo o tiempo que requiere para llevarse a cabo un proyecto. Cabe mencionar que dichos cálculos son aproximaciones al esfuerzo real, debido a que es muy difícil, por la misma naturaleza de los proyectos de desarrollo de software, realizar un cálculo exacto.

A2.1 Estimación tradicional

Entre las medidas de estimación tradicionales se pueden enunciar los muy conocidos Puntos de función y la de Líneas de código que de ahora en adelante denominaremos PF y LOC (Line Of Code) respectivamente. La medida de estimación de LOC tiene como propósito definir el tiempo y el costo del proyecto con base a la cantidad de líneas de código que se tienen que escribir durante el desarrollo, pudiéndose determinar así, el costo por línea y cuantas líneas de código se desarrollan por mes. Pero LOC tiene ciertas dificultades entre las cuales se podrían nombrar las siguientes:

- El número de LOC dependerá en gran parte de la habilidad y destreza del programador que lo esté realizando.
- Existe una relación inversa entre el nivel del lenguaje (lenguaje de alto nivel o lenguaje ensamblador) y la producción de trabajo.
- El número real de LOC no se sabrá hasta que el desarrollo del software esté casi terminado.
- No hay un método aceptado para contar líneas de código, aunque existen diversas aplicaciones que contribuyen a la realización de esta labor, entre las cuales se podrían nombrar: CodeCount, CCCC, K-LOC Calculator, EZMetrix, Code Analyzer, solo por mencionar algunos.

Por otra parte, los PF son utilizados para la estimación en función de los atributos del sistema a desarrollar. Para realizar el cálculo basado en PF se deben tener en cuenta las siguientes propiedades:

- Número de entradas de usuario
- Número de salidas del usuario
- Número de consultas de los usuarios
- Número de archivos
- Número de interfaces externas

Una vez que estas propiedades han sido colocadas en la tabla se elige un peso, el cual representará el grado de complejidad que se prevé para cada una de las opciones. El cálculo de los PF se realizará entonces basado en la siguiente fórmula:

$$PF = Cuent \quad Total * [0.65 + 0.01 * SUM(f1)]$$

donde la variable función SUM (f1), será el resultado de la sumatoria de aplicación de las siguientes preguntas asociadas a su respectiva puntuación en el rango de 0-5 (0 no influyente y 5 esencial)

1. ¿Requiere el sistema copias de seguridad y de recuperación fiables?
2. ¿Se requieren comunicaciones de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es crítico el rendimiento?
5. ¿Será ejecuta el programa en un sistema operativo existente y fuertemente utilizado?
6. ¿Requiere el sistema entrada de datos interactiva?
7. ¿Requiere la entrada de datos interactiva que se utilicen varias pantallas o varias operaciones?
8. ¿Se utilizan los archivos maestros de forma interactiva?
9. ¿Son complejas las entradas, las salidas y/o las peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidas en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar diferentes instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Los PF los podemos utilizar para calcular:

- Productividad = PF/Personas-mes
- Calidad = Errores/PF
- Costo = Pesos/PF
- Documentación = Páginas /PF

A2.2 Estimación en proyectos ágiles

Las metodologías ágiles deben ser adaptables a las necesidades cambiantes en cualquier momento durante la vida del proyecto, a diferencia del enfoque tradicional que tratan de definir todos los requisitos en el inicio del proyecto y luego ampliar los esfuerzos para controlar los cambios a los requisitos. Es por esto que las metodologías Ágiles trabajan bajo el principio de entrega de una salida de trabajo, reunir otros requisitos y evolucionar el producto.

➢ Puntos de historia

Se podría afirmar que los puntos de historia son una medida de complejidad y esfuerzo. Algunos equipos optan por tomarlos como medida de complejidad, pero estos también se pueden orientar para estimar el esfuerzo requerido para desarrollar un producto, incluyendo los riesgos y la incertidumbre. Definamos entonces un punto de historia a partir de la siguiente ecuación:

$$\boldsymbol{Punto\ de\ historia} = Esfuerzo + Complejidad + R$$

Analicemos el siguiente contexto: Supongamos que alguien quiere ir a su oficina. ¿Qué medio de transporte usaría para llegar? Puede que esta persona decida tomar un taxi o un autobús dependiendo del tráfico. ¿Cuánto tiempo se tardaría en ir de casa a la oficina? Multitud de otros factores variables influirán de una forma u otra en el resultado final que debe ser, en este caso, llegar a la oficina. Por otro lado, si ahora nos preguntamos "¿Cuál es la distancia entre la casa y la oficina?", la respuesta a este interrogante será una constante. Lo cual nos lleva a afirmar que "No importa cuál sea el medio de transporte que se use o el tráfico, la distancia entre la casa y la oficina sigue siendo la misma." Bueno, ahora pensemos en los puntos de historia, estos son como la estimación de la distancia. Ellos siguen siendo los mismos, sin importar quién está trabajando en la historia de usuario. El tiempo puede variar en función de muchos factores: La persona que trabaja en la historia, el tiempo de reunión, tiempo perdido, todo contribuye al tiempo real invertido, pero el tamaño sigue siendo el mismo. Los puntos de historia son relativos, es decir, supongamos que durante el desarrollo de un proyecto se elige una historia de usuario como referencia y se le asigna un valor de 1 punto. A continuación, se debe medir cada historia en relación con ésta, de tal forma que a medida en que aumenta la complejidad así aumentará el valor del punto. Los puntos de historia no intentan asignar la duración de cada historia, sino que permiten una comparación entre las tareas.

Las historias de usuario deben ser:

- Comprensibles para los clientes y desarrolladores.
- Comprobables.

- Del tamaño justo. Se recomienda que sean pequeñas como para que el programador pueda crear varias en una iteración.

➢ Estimación en puntos de historia

Los puntos de historia indican el tamaño y la complejidad dada una historia de usuario con relación a otras historias que son parte del proyecto. Determinar el número de puntos de historia en cada historia de usuario es subjetivo. Los puntos de historia permiten estimar esfuerzo sin tratar de estimar cuanto tiempo tomará. Ahora si nos cuestionamos sobre ¿Cuánto tiempo se tarda en desarrollar un software?, la respuesta sería: se suman los puntos de historia y luego dividimos entre la velocidad, y eso nos daría el tiempo:

$$\boldsymbol{Velocidad\ del\ equipo} = \frac{Puntos\ de\ historia}{Iteración}$$

¿Pero cómo de grande es un punto de historia? ¿Y cómo puedo saber qué es la velocidad? Si estamos trabajando bajo el enfoque de Scrum, las estimaciones de los puntos de historia son realizadas por el equipo que realiza el desarrollo. Mirándolo desde esta perspectiva, son ellos los que definen un punto de historia. Algunos desarrolladores estiman los puntos de historia basados en la experiencia y conocimientos, a partir de ahí, hacen una estimación del coste que supondrá la realización de esa tarea; es así como son asignados los niveles de los puntos de historia. Esta es la forma más sencilla para estimar historias de usuario, referenciándose a una historia similar producto de la experiencia. Otros lo estiman leyendo la lista de tareas a realizar, toman lo que parece ser lo más sencillo, basados en la experiencia, y le asignan el valor de 2. Y a partir de allí, todo lo demás se estima en relación con esa tarea. Pero si analizamos lo anterior, nos damos cuenta de que estimar de esta forma puede llegar a ser muy impreciso. Más formalmente, los puntos de historia se calculan por lo general en la escala de Cohn, denominado de esta forma por Mike Cohn, esta es la escala de estimación:

0, 1, 2, 3, 5, 8, 13, 20, 40, 100.

Ahora surge una duda: ¿Cómo calculamos la velocidad? En el primer sprint, el equipo acepta el trabajo basados en la experiencia y el debate. Después de cada Sprint, se mide la velocidad. Después de dos o tres sprints, la velocidad media de medición pueden ser utilizados para predecir la velocidad en el futuro, y por lo tanto la fecha de finalización

➢ Técnicas de estimación - Planning Poker

Planning Poker, es una técnica de estimación para proyectos ágiles propuesta en un inicio por Grenning en el año 2002 y popularizada por Cohn, la cual permite hacer una estimación inicial rápida del proyecto. Esta estimación se realiza de manera consensuada con base en los requisitos.

El proceso de Planning poker se inicia con una reunión entre los miembros del equipo, se plantea una historia de usuario, se analiza y utilizando las cartas cada miembro muestra cuanto cree que sea el valor que debe asignarse. Como número resultante se puede tomar la más alta, la media, la más baja, o cualquier otro método. El valor asignado debe ser calculado con un método consensuado por todos los miembros del equipo. Una buen método que puede facilitar la tarea de consenso es preguntar al equipo el por qué de determinados valores, después de escuchar las razones se propone volver a votar.

Para estimar por medio de esta técnica necesitaremos:

- Una lista de características, basada en las historias de usuario, la cual describirá el software a desarrollar.
- Una baraja de cartas. Existen dos tipos de barajas. Una contiene cartas con la secuencia de Fibonacci incluyendo un cero:

0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

Otros contienen progresiones similares. La razón de utilizar la secuencia de Fibonacci es reflejar el incierto inherente en la estimación. Sin embargo, la baraja usada comúnmente es la secuencia:

0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100.

La baraja también trae incluida opcionalmente, cartas con los símbolos de “?”, que significa inseguridad o duda y una taza de café, que hace referencia a la necesidad de un break o descanso. La carta con el cero (0) significa que esta historia de usuario ya ha sido realizada.

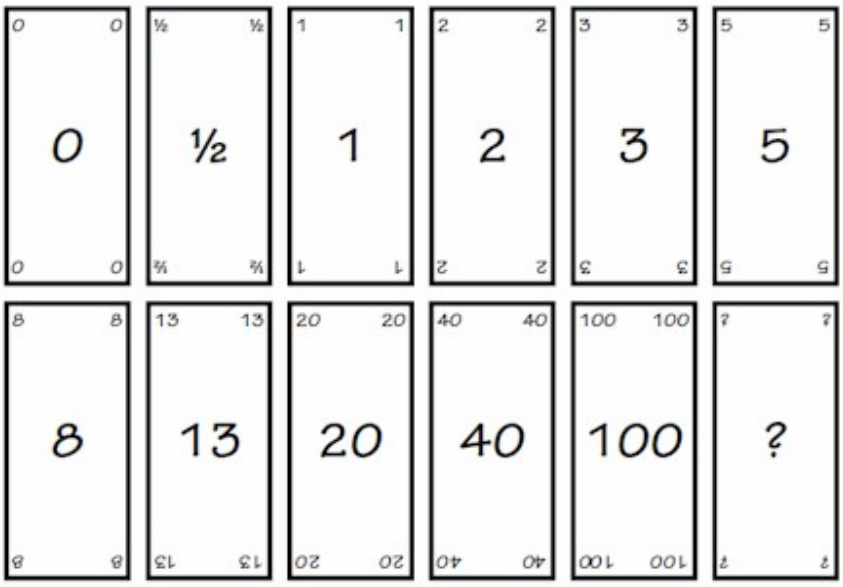


Figura 71. Ejemplo de baraja de planning póker

Glosario de términos

- **Agilismo**: Término que se usa en lugar del término anglosajón "Agile". Se pueden utilizar los términos desarrollo ágil de software, metodología ágil ó agilidad.
- **Gráfica Burndown:** Gráfica que muestra la velocidad a la que se están completando los requisitos del sprint.
- **Historia de usuario:** En las metodologías ágiles este término sustituye a la especificación de requisitos. Las historias de usuario son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos.
- **Kanban**: Método ágil para la gestión de desarrollo software.
- **Manifiesto ágil**: Recoge los principios en los que se basan todas las metodologías ágiles.
- **Metodología ágil:** Métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requerimientos y soluciones evolucionan mediante la colaboración de grupos auto-organizados y multidisciplinarios flexibles al cambio.
- **Product backlog**: Lista de requisitos priorizada por el cliente.
- **Product owner:** En equipos ágiles representa al cliente y es el encargado de negociar con el equipo.
- **QA**: Del anglosajón Quality Assurance, son las tareas, previas a la entrega al cliente, que se llevan a cabo para asegurar la calidad del producto software.
- **Refactorizar**: Técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.
- **Scrum**: Método ágil para la gestión de desarrollo software.

- **Scrumban**: Método ágil para la gestión de desarrollo software.
- **Scrum daily meeting:** Reunión diaria de Scrum en la que participan todos los miembros del equipo y comentan el progreso del trabajo y posibles bloqueos que hayan surgido en el trabajo.
- **Scrum master**: En equipos ágiles responsable de guiar y resolver obstáculos al equipo.
- **Sprint o iteración**: Ritmo de los ciclos de Scrum. Está delimitado por la reunión de planificación del sprint y la reunión retrospectiva.
- **Sprint backlog**: Lista priorizada con los requisitos seleccionados para un Sprint.
- **Sprint demonstration o demo:** Reunión con el cliente, propia de Scrum, en la que se muestran las nuevas funcionalidades.
- **Sprint restrospective o retrospectiva**: Reunión propia de Scrum, en la que se reflexiona sobre el Sprint que acaba de finalizar y se determina qué podría cambiarse para el siguiente Sprint y ser más productivo y mejor.
- **Velocidad del equipo o velocidad de trabajo**: Número de puntos ideales de trabajo que es capaz de asumir el equipo en el presente Sprint.
- **XP:** Método ágil para la gestión de desarrollo software.

Referencias

[1] PROJECT MANAGEMENT INSTITUTE. A Guide to the Project Management Body of Knowledge – PMBOK® Guide Fifth Edition.

[2] A. H.-S. B. Qumer, «An evaluation of the degree of agility in six agile methods and its applicability for method engineering.,» 2007.

[3] K. Beck+, «Manifesto for Agile Software Development,» 2001.

[4] «aegxxi-desarrollo,» [En línea]. Available: http://aegxxi-desarrollo.blogspot.com.es/2012_06_01_archive.html. [Último acceso: Septiembre 2015].

[5] O. d. l. Cuesta, «Palentino Blog,» [En línea]. Available: http://www.palentino.es/blog/resumen-de-la-metodologia-scrum-para-el-desarrollo-del-software-historia-caracteristicas-roles/. [Último acceso: Agosto 2015].

[6] «Mountain Goat Software,» [En línea]. Available: http://www.mountaingoatsoftware.com/agile/scrum/task-boards. [Último acceso: Septiembre 2015].

[7] M. Fowler, Refactoring: Improving the Design of Existing Code, 2002.

[8] «Tras la niebla,» [En línea]. Available: http://www.traslaniebla.com/2014/03/20/organizate-con-kanban/. [Último acceso: Septiembre 2015].

[9] «Net Objetives,» [En línea]. Available: http://www.netobjectives.com/. [Último acceso: Septiembre 2015].

[10] A. I. a. C. Souveyet, «Framework for Agile Methods Classification,» 2008.

[11] A. I. a. C. Souveye, «Framework for Agile Methods Classification».

[12] A. M. I. Solano, «Importance of Story Points in the Estimation of Projects Under the Agile Approach,» 2011.

[13] J. D. F. M. J. M. V. Andrés Navarro Cadavid, «A review of agile methodologies for software development,» 2013.

[14] J. Palacio, Scrum Manager I - Las reglas de scrum, 2014.

[15] N. &. Tekeuchi, The New New Product Development Game, 1986.

[16] M. J. P. Pérez, Guía Comparativa de Metodologías Ágiles.

[17] «agiletropic,» [En línea]. Available: http://www.agiletropic.com/.

[18] J. Palacio, «Origen de la gestión de proyectos,» 2006. [En línea]. Available: http://www.navegapolis.net. [Último acceso: Mayo 2015].

[19] J. Garzas. [En línea]. Available: http://www.javiergarzas.com/2012/09/metodologias-crystal.html. [Último acceso: Junio 2015].

[20] «PMO Informática,» [En línea]. Available: http://www.pmoinformatica.com/. [Último acceso: Junio 2015].